DREAMS and IMAGE: A Model and Computer Implementation for Concurrent, Life-Cycle Design of Complex Systems

Mark A. Hale, James I. Craig, Farrokh Mistree, Daniel P. Schrage

Aerospace Systems Design and Systems Realization Laboratories
Georgia Institute of Technology, Atlanta, Georgia 30332

Mark A. Hale
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0150
(404) 894-9810
(404) 894-9812 (FAX)

James I. Craig
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0150
(404) 894-3042
(404) 894-2760 (FAX)

Corresponding Author:
Farrokh Mistree
School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0405
(404) 894-8412
(404) 894-9342 (FAX)

Daniel P. Schrage
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0150
(404) 894-6257
(404) 894-2760 (FAX)

Hale, M. A., J. I. Craig, F. Mistree and D. P. Schrage, "DREAMS & IMAGE: A Model and Computer Implementation for Concurrent, Life-Cycle Design of Complex Systems," Concurrent Engineering: Research and Applications, vol. 4, no. 2, pp. 171-186,

Abstract: Computing architectures are being assembled that extend concurrent engineering practices by providing more efficient execution and collaboration on distributed, heterogeneous computing networks. Built on the successes of initial architectures, requirements for a next-generation design computing infrastructure can be developed. These requirements concentrate on those needed by a designer in decision-making processes from product conception to recycling and can be categorized in two areas: design process and design information management. A designer both designs and executes design processes throughout design time to achieve better product and process capabilities while expending fewer resources. In order to accomplish this, information, or more appropriately design knowledge, needs to be adequately managed during product and process decomposition as well as recomposition. A foundation has been laid that captures these requirements in a design architecture called DREAMS (**D**eveloping **R**obust **E**ngineering **A**nalysis **M**odels and **S**pecifications). In addition, a computing infrastructure, called IMAGE (**I**ntelligent **M**ultidisciplinary **A**ircraft **G**eneration **E**nvironment), is being developed that satisfies design requirements defined in DREAMS and incorporates enabling computational technologies.

Keywords: life-cycle design, decision-making, design processes, Decision-Based Design, computer architecture, information, database, schema, agent

Glossary

Design Accuracy	A measure of the correctness of a design.
Agent	A resource that has been modeled and wrapped for inclusion in a distributed design environment.
DBD **D**ecision-**B**ased **D**esign	A paradigm that captures the notion that the principal role of a designer is to make decisions.
DREAMS **D**eveloping **R**obust **E**ngineering **A**nalysis **M**odels and **S**pecifications	An architecture that formally supports Decision-Based Design.
DSPT **D**ecision **S**upport **P**roblem **T**echnique	A technique for implementing Integrated Product and Process Development from a decision-based perspective. The technique facilitates meta-design, the partitioning of a design problem through the use of Support Problems, and actual design, the solution of Support Problems.
Design Fidelity	A measure of the completeness of a design.
IMAGE **I**ntelligent **M**ultidisciplinary **A**ircraft **G**eneration **E**nvironment	A computing infrastructure that facilitates IPPD from a decision-based perspective.
IPPD **I**ntegrated **P**roduct and **P**rocess **D**evelopment	Embodies the simultaneous application of both system and quality engineering methods throughout an iterative design process.
NII **N**ational **I**nformation **I**nfrastructure	A conglomeration of users and services on the internet.
Schema	A collection of attributes and their instances.
Support Problem	Support Problems govern the transformation of information into knowledge. Support Problems have a structure defined by keywords and can be used to model an entire design timeline.

1. Introduction

A number of preliminary computational frameworks have been assembled and demonstrate the computational technologies required for computing resource integration.[1-11] The development of these frameworks have been driven by computational demonstrations rather than those imposed by a designer performing the concurrent design of engineering systems. In the context of this paper, the term "a designer" would also include geographically distributed **I**ntegrated **P**roduct **T**eams (**IPT**). This distinction is important because of the additional design data and process complexity imposed by IPT's operating in this manner. The successful use of a computational architecture requires that the architecture be derived from those activities that occur during concurrent engineering and incorporates the technologies found in previous architectures.

The authors have attacked this problem by defining requirements from a designer-centered approach for defining design activities. The result of this approach is requirements that are defined by the need for a design partitioning and solution scheme as well as a suitable information model. It will be shown that a paradigm shift, from optimal to satisficing, will be required in terms of the

solution techniques that are applied at particular points along a design timeline. A generic design implementation, called DREAMS, will be described that satisfies the needs of a designer. IMAGE is a computational infrastructure that is an implementation of the DREAMS architecture. This revolutionary infrastructure combines the requirements needed to support concurrent design activities as well as integrates existing as well as newer enabling computational technologies.

2. A Design Timeline

As process information is encountered earlier and design information is streamlined, a designer performs many design activities during the simultaneous design of engineering systems. These processes are encountered throughout a product's life-cycle, from conception to retirement/recycle. During these processes, a designer is concerned with:

- Customer Requirements,
- System Affordability and Robustness,
- Open Systems, and
- Resource Expenditure.

These concerns can be summarized with the expression "a designer would like to do more with less". A design architecture is required to provide the flexibility needed by a designer to address these concerns during system partitioning and solution.

One way of modeling processes involved in system partitioning is through the use of the **D**ecision **S**upport **P**roblem (DSP) Technique. The DSP Technique provides a set of base entities that are used by a designer to model a design timeline. These entities are systematically arranged into multi-level networks that represent design processes. In turn, these networks are solved in order to generate design information for decision-making. This process is described in more detail in discussions to follow. A designer may describe current design processes based on legacy design models (prescriptive models) or create new design models (descriptive models) and then proceed to solve the design problem.[12]

There are two phases encountered by a designer when employing the DSP Technique:

- **Meta-design**: whereby a designer partitions a design timeline with the aid of Support Problems; and
- **Actual design**, whereby Support Problems are exercised so that knowledge about a design can be generated and decisions can be made.

During the meta-design phase, a designer explicitly models design processes using Support Problems. Several activities that may occur during a generic design process are shown in Figure 1. The activities are depicted in a manner consistent with a traditional design timeline. A designer can use the DSP Technique to partition this design timeline into Support Problems. For

example, the timeline from Figure 1 can be partitioned as shown in Figure 2. Design Phases[1] and Events[2] are identified by a designer during the partitioning process. A designer also identifies information that is to be accumulated for making decisions. For the generic case represented in Figure 2, a designer has identified four design Phases and three design Events that occur during the *Conceptual Design* Phase. Realizing the impetus of IPPD, the DSP Technique allows for Design Phases and Events to occur simultaneously.

[Figures 1 & 2 go here]

Support Problems may be formulated that represent design activities that are to occur during design processes. A partial representation of the Support Problems that comprise the shaded timeline partition in Figure 2 are shown in Figure 3. This schematic depicts the transformation of the *Statement of Requirements* into *Top Level Specifications* through two design Phases and three design Events. Notice that the schematic is multi-level. The lower sequence represents the activities that comprise the *Conceptual Design* Phase. Design Events may occur simultaneously, thus requiring the use of multidisciplinary and concurrent analysis techniques. Transmission Entities, shown as Information blocks with a diamond-shaped border (see Table 1), encapsulate the respective information requirements that are shared among Support Problems. After the Phases and Events of Figure 3 have been configured into a network, a designer would go on to determine the underlying networks that describe more detailed design processes.

[Figure 3 and Table 1 go here]

A designer would like to use a computer-based architecture to facilitate activities that occur during design processes, namely design timeline partitioning and solution. This architecture integrates legacy analysis programs with dynamic process modeling techniques. Repetitive and monogamous tasks of program execution and data archiving can be automated. Thus, a designer is free to focus more on decision-making assisted by visualization tools. The following statements describe the functionality required of such an architecture:

- A designer performs design activities to generate design knowledge for effective and efficient decision-making;
- A design timeline is partitioned into manageable units;
- A partitioning scheme accurately and consistently reflects design activities; and
- A means exists for solving the partitioned design timeline.

---

1 Design Phases are distinct stages of a design's development whose boundaries are usually characterized by changes in design fidelity.

2 A Design Event is a coincidence of design activities that are to occur so that the solution of a problem or sub-problem can be achieved.

If Support Problems are to be used to partition a design timeline, they must present a consistent model for transforming information into knowledge throughout a design's life-cycle. A consistent model will be presented in the next section based on the notion of "satisficing" solutions.[3]

3. Design Freedom Through Satisficing Solutions

Support Problems are applied throughout a design timeline; therefore, their nature must be such that they can be consistently applied during a design process. Before discussing a model for Support Problems, the following are noted about activities that occur throughout design processes:

- It is desired to minimize the impact of early design decisions on product changes, variance, and openness through robust design techniques.
- Design costs are locked in early in the design process as design decisions are made.
- It is desired to minimize risk through product robustness.
- As a design progresses, more optimal solutions (design knowledge) may be obtained since more is known and a design becomes less ambiguous.
- Design decisions that are made with respect to decomposed design problems must prove to be equally as valid with respect to a recomposed system that may not be assembled until later in a design process.
- For informed decision-making, a designer desires as much accountable design knowledge that can be consistently drawn from design models.
- In the early stages of design, decisions are made with respect to low-fidelity, less-accurate representations that considerably impact a design. This idea is captured in the familiar design freedom/knowledge versus time curves shown in Figure 4: design freedom is locked in when little is known about a design. In addition, cost is determined and risk is assumed.

The use of Support Problems provide a means of supporting these activities. Support Problems encapsulate methods, arranged as networks, that are used to generate design information for decision-making. As a result, design processes can be represented by a cascade of Support Problem networks. The different types of Support Problems used by a designer to model a design timeline and their theoretical basis are described below.

[Figure 4 goes here]

Support Problems that are employed by a designer at the first or highest level of design activities that may occur are illustrated in Figure 2. As design processes are decomposed further, other Support Problems may be encountered. The different types of Support Problems and their

---

[3] A satisficing solution is one that provides a region of solutions that minimizes the deviation between customer and manufacturer requirements and design constraints, bounds, and goals.

corresponding function are outlined in Table 1. Using these Support Problems, design processes can be represented as networks and solved.

The fundamental notion behind the use of Support Problems is as follows. Traditionally, decisions have been based on optimality criteria imposed locally on a limited design representation. As designs progress, either local or system level changes may cause an optimal target to shift, rendering the design infeasible. The ideas behind optimal solutions are depicted in Figure 5. Initially, a system may optimally satisfy problem constraints and customer requirements, represented by peaks in the solution space shown in Figure 5. A particular problem solution is represented by a ball in the Figure. A problem shift will cause the system to deviate from an optimal solution, thus rendering the initial solution to be sub-optimal or even infeasible.

[Figures 5 goes here]

Early in a design process, a satisficing model is more appropriate since the model presents a robust, open, system-oriented solution for systems with a high degree of ambiguity. The use of satisficing solutions during initial design processes is a fundamental paradigm shift from traditional, optimal approaches. However, such a shift is required if design processes are to be represented across a design timeline and suitable design information requirements are to be derived.

A satisficing solution is one that provides a region of solutions that minimizes the deviation between customer and manufacturer requirements and design constraints, bounds, and goals, see Figure 6. As a result, a designer can base decisions about a design on regions of plausible design derivatives/alternatives that exist at that point on the design timeline. Moreover, a designer can make more confident decisions than those made at the subsystem level. A pictorial aid for the notion of satisficing solutions is presented in Figure 7. Optimal peaks are replaced by satisficing mesas, leading to robust design solution regions. Early in the design process, a designer bases decisions on a region of acceptable design solutions. As the region evolves throughout design processes, particular design decisions remain valid and lie within the region of candidate solutions (a mesa).

[Figures 6 & 7 go here]

These two models, satisficing and optimal, are encountered as Support Problems are used in design processes. As shown in Figure 8, satisficing solutions are used early in design processes since less is known about designs. Represented by a fading timeline, the need and use of satisficing solutions diminishes as a design progresses. Notice also that the timeline has been partitioned, as shown by the parallelograms. This represents a timeline that has been appropriately

modeled with Support Problem networks and allows for the use of different solution techniques. As designs are refined, more is known about a design and a designer begins to look for solutions that approach optimal type solutions, as seen in Figure 9. At this point, traditional optimization methods as well as newer global sensitivity approaches may be used to aid in problem solution.[13, 14]

[Figures 8 & 9 go here]

As designs are partitioned, a designer employes the use of Support Problems that adhere to this notion of satisficing solutions. These Support Problems are word formulated using a standard template. In the case of a Compromise Decision Support Problem (see Figure 6) , the template contains the keywords Given, Find, Satisfy, Minimize. From this template, various equations and algorithms are identified and a solutions is obtained. Later, a systematic approach to formulating and solving Support Problems will be described.

4. Design Information Requirements

Having examined the requirements imposed on a design computing architecture by design timeline partitioning needs, attention is focused on the underlying information model. **O**bject-**O**riented **D**atabase **M**anagement **S**ystems (**OODBMS**) are currently being developed or are already in place for information storage. These systems provide facilities for object creation, revision, distribution, storage/retrieval, and method definition. There are a number of commercial systems that make use of distributed methods in heterogeneous computing networks. These systems provide models for object definitions and relationships. However, these models do not reflect the behavior required for a system to manipulate design information.

Earlier, it was proposed that one way to represent design processes would be to decompose those processes into Support Problems as elementary units. Information requirements arise from the use Support Problems a design partitioning scheme as well as object-oriented models:

- Information is both structured and unstructured.
- A design encompasses both form and function;
- A design decomposition is not necessarily unique;
- Design fidelity (degree of completeness) increases as a design progresses;
- A design may be represented by varying levels of accuracy at a given level of fidelity;
- Informed decision-making requires that knowledge be used in context; and
- The information model must be dynamic so that models may evolve with a design timeline.

A computer architecture will need to provide comprehensive information management that satisfies these requirements.

5. An Information Model

Based on the dynamic information requirements described above, a generic design management scheme can be developed that satisfies them. As represented by the icons in Figure 10, information can be either structured (an information hierarchy) or unstructured (an information heterarchy). The information heterarchy refers to unstructured information, or loose information.[15] During design processes, some information will be structured from the heterarchy into the information hierarchy. An example of unstructured information would include local program variables and unexplored design variants. Heterarchical information can be represented by a disjoint collection of objects as shown in Figure 10b.

[Figures 10 goes here]

Stephens has shown that design hierarchies can be used to represent a design space. A design space consists of all of the temporal information encountered during design. This includes physical product models (eg CAD geometry), process flow diagrams, product metrics, descriptions of analysis programs, legacy data, and so forth. Notice that a design space includes more than a description of a physical artifact. It also contains product functionality, solution modeling, and the process of designing itself. The Form-Function-Process-Model/Temporal design sub-spaces capture this information in a design space and are summarized in Table 2.[16] The four sub-spaces, Form, Function, Process, and Model, are populated by multiply-connected, and therefore two-dimensional, object hierarchies. These hierarchies are 2D because the hierarchies can be represented by simple object-based, parent-child relationships. For example, a design space has been partitioned into these elements and is shown in Figure 10a. Furthermore, the Form sub-space may be populated by the objects and organized into the hierarchy shown in Table 3. These four sub-spaces along with time span nine-dimensions as given in Table 2.

[Tables 2 & 3 go here]

Looking again at Figure 10a, Process elements are formed by a Form-Function-Model triplet. Later, it will be shown that Process elements can be implemented on the computer as software Agents. Because Process elements explicitly contain Models, these Agents can be used by a designer to produce design information in context. Context is provided because Agents are scriptable and they publish their capabilities and activities. The mechanism by which this occurs will be discussed in more detail in the implementation section. Design information provided in context is referred to as knowledge. Therefore, a designer may interrogate design information to

determine who produced it, when it was created, what was used to produce it, etc. Thus, accountable design information may be obtained through the use of Agents.

As designs progress, the Information model must support increasing levels of fidelity consistent with increasing completeness of a design. A `Lifting Surface` Form object described to a tertiary level of fidelity from a structural standpoint is shown in Table 4. Each object encapsulates one or more schemas, each potentially having different degrees of accuracy. Three schemas that may be used to represent a `Lifting Surface` Form object in conceptual design are illustrated in Table 5. The ability to support increasing levels of accuracy within the information model is called schema evolution. The inability to support schema evolution because they use static classes is the main weakness of present day OODBMS. However, schema evolution can be implemented on top of these systems by having objects manage linked-lists of schemas. In turn, schemas manage linked-lists of attributes. This approach is inherently dynamic but at a time expense.

[Tables 4 & 5 go here]

6. DREAMS - A Design Process and Information Architecture

A generic architecture has been developed that incorporates the design process as well as information management schemes discussed earlier. This architecture is shown in Figure 11. The architecture has three fundamental components:

- **System Partitioning and Execution**: Facilities are provided that permit a designer to partition a design problem into manageable sub-problems and for the solution of the resulting sub-problems;
- **Support Problem Definition and Solution**: A consistent scheme has been developed that allows a designer to define and solve sub-problems, which are called Support Problems; and
- **Design Management**: Utilities are provided to a designer that allow for comprehensive, object-oriented data management throughout design processes.

The interdependency between Design Processes and Support Problem Definition and Solution is indicated by circular arrows. Formulation, Translation, and Evaluation are encountered as meta and actual design activities occur. During all Design Processes, information exchange takes place and is indicated in by horizontal, bi-directional arrows between Support Problems in their intermediate stages and a Design Specification. Utilizing these three components, a designer can manage complex system design problems throughout all design processes. This process

architecture has been given the acronym DREAMS (**D**eveloping **R**obust **E**ngineering **A**nalysis **M**odels and **S**pecifications). Each of the three components will be discussed in turn.

[Figure 11 goes here]

6.1 System Partitioning through the DSP Technique

The processes involved in system partitioning can be modeled using a DSPT Palette.[12] A DSPT Palette provides a set of base entities that can be used by a designer to model a design timeline. A designer can use legacy design models (prescriptive models) or create new design models (descriptive models) to describe current design processes and then proceed to solve the design problem. The DSPT Palette implementation provides an interface for meta-design activities (partitioning a design timeline) and actual designing (solving Support Problems). The DSPT Palette supports the following functionality:

- A top-level Design Palette for multiple problem management;
- Additional Palettes that allow for problem decomposition;
- Information recomposition from lower Palettes to higher Palettes; and
- Multiple-user problem definition.

These functions allow a designer or an Integrated Product Team to decompose and solve a design problem throughout a design timeline. After defining design processes in meta-design, a designer can use the DSP Technique to generate knowledge used for decision-making by solving the resulting Support Problems. For example, each icon in Figure 3 corresponds to an associated Support Problem. A standard technique exists for describing and solving Support Problems and is outlined in the next section.

6.2 Support Problem Definition and Solution

Within the DSP Technique, Support Problems are exercised by a designer to produce knowledge about a design so that decisions can be made based on that knowledge. Support Problems provide standard models for transforming design information into knowledge. There are three steps required in defining and solving Support Problems:

- **Formulation**: The structuring of the problem statement into specific Support Problem models;
- **Translation**: Associating processes, that govern the generation of information into knowledge, with the Support Problems; and
- **Evaluation**: Producing design knowledge through the solution of the Support Problems.

These steps will be explained and illustrated in the sections to follow.

6.2.1 Formulation - Support Problem

Support Problems are defined when a design process is partitioned in meta-design. Support Problems have a defined structure given by keywords, see Figure 6. The Compromise DSP has the following form:[15]

Given: Feasible design and aspiration space
Find: Values of variables
Satisfy: System constraints, bounds, and goals
Minimize: Deviation between "what I want" and "what I can have"

Support Problems are formulated as linguistic statements stemming from design problem statements, customer desirements, and brainstorming. As Support Problems are formulated, they are embodied by Forms and Functions from a Design Specification. The formulation of a multidisciplinary wing integration Compromise Decision Support Problem is illustrated in Figure 12. As shown in Figure 13, the definition of Support Problems begins with a Compromise DSP. As Support Problems definitions are completed, knowledge about Problems increases. This can be visualized as a cone of increasing girth.

[Figures 12 & 13 go here]

6.2.2 Translation - Math Form

Once a Support Problem has been formulated, the problem is then translated into an equivalent Math Form. The Math Form for Phases, Events, Tasks, and Decisions are Support Problem networks. These networks can be created automatically by using a natural language processor to parse the linguistically formulated Support Problem. In the case of System Support Problems, the Math Form provides the process connectivity between Forms and Functions. In this case, a designer manually selects a suitable Form-Function-Model triplet. This triplet is a Process element. For instance, the functions `Lift` and `Drag` are associated with the form `Wing` through the relations:

$$Lift = C_l \bullet q \bullet S \tag{1}$$

$$Drag = C_d \bullet q \bullet S \tag{2}$$

where C_l and C_d are the respective lift and drag coefficients, q is the dynamic pressure, and S is the wing area. As the Math Form becomes more complex, equations are typically grouped into engineering models. In turn, models are often grouped into disciplines. Some of the traditional aerospace disciplines that are present in the multidisciplinary wing integration problem are shown in Figure 12. Notice that inter-disciplinary models do exist, as in the case of aeroelasticity, and must be accounted for. The Math Form adds additional information to an existing Compromise DSP definition as shown in Figure 13.

6.2.3 Evaluation - Template

Finally, the Math Form of the Support Problem can be solved. The Support Problem solution consists of three steps: pairing the Math Form with a suitable Agent, structuring a solution network, and solving the Problem. Agents are used by a designer to generate design information from the expressions found in the Math Form of a Support Problem[4]. As shown in Figure 12, Agents are typically engineering analysis codes. Other Agents include expert systems, hyper-media sources, virtual reality, and the human designer. After the Math Form and Agents have been collected, the new form of the Support Problem is called the Support Problem Template. A Template completes a Support Problems definition and is pictorially represented as the base of the Support Problem cone shown in Figure 13. The notion of the SP Template is important for modeling design processes. SP Templates represent a bridge between modeling design processes and systematically employing Available Assets to generate the information required for those processes.

Continuing, a solution network must be formed detailing the sequence of events required for solution. DeMAID is a tool that assists in the ordering of analysis modules for execution.[17] Design Structure Matrices may be ordered with respect to feedback loops, time, and cost considerations. Stettner has added to DeMAID by distinguishing both sequences and circuits as necessary components of MDO activities.[18] Moreover, Stettner has outlined a decomposition and recomposition method for forming sequences and circuits.

Finally, the Support Problem must be solved. **D**ecision **S**upport **I**n the **D**esign of **E**ngineering **S**ystems (DSIDES) is a suite of tools used to solve Support Problems.[19] Tools in DSIDES are used to solve Selection DSPs (SELECT) and multi-level, multi-goal Compromise DSPs (ALP).

6.3 Design Specification and Heterarchy

The Design Specification Editor is a tool for comprehensive information management and implements the notions of both information heterarchies and hierarchies. The Editor is based on an earlier information system called DEFINE implemented as part of a **L**aboratory **E**nvironment for the **G**eneration, **E**valuation, and **N**avigation of **D**esign (LEGEND).[16] The Editor includes:

- Generation of Form, Function, Process, and Model hierarchies;
- Heterarchical to hierarchical object migration;
- Varying levels of fidelity;
- Schema development and evolution;
- Instance accumulation provided in context;
- Object sharing among projects and team members; and

---

4 Agents will be discussed in more detail in the computer implementation sections to follow.

- An interpretive object-oriented database management system including inheritance, persistence, memory management, and object condensation.

These capabilities allow for comprehensive design information management based on a common product data model.

7. IMAGE

A revolutionary computing infrastructure has been implemented based on the requirements outlined earlier for the concurrent design of engineering systems. Some unique and distinguishing features of this architecture are:

- A design partitioning process;
- A mechanism for solving Support Problems;
- A design information model;
- Information generation in context for informed decision-making;
- Efficient and cost-effective application of design resources; and
- Geographically distributed design activities.

The resulting infrastructure is called IMAGE, an **I**ntelligent **M**ultidisciplinary **A**ircraft **G**eneration **E**nvironment.

A diagram of the IMAGE infrastructure is shown in Figure 14. This infrastructure is comparable to the **F**ramework for **I**nterdisciplinary **D**esign **O**ptimization (FIDO), **A**ffordable **S**ystems **O**ptimization **P**rocess (ASOP), and other efforts in the development of underlying computing technologies.[11, 20] However, the IMAGE infrastructure is designed to explicitly support general design activities and an information model within an accountable design context. This explicit support for design of complex engineering systems is a notable enhancement of this architecture over others. These design requirements dictate the implementation strengths of the synergistic information and process model described earlier.

[Figure 14 goes here]

As shown in Figure 14, IMAGE has the following features:

- **Designer Activities:** A designer partitions a problem into activities for solution as well as to provide comprehensive information management;
- **Available Assets:** A variety of design resources are provided to aid in the generation of design knowledge. Resources range from object-oriented databases to CAD packages;
- **Agent Collaboration** (indicated by Models and Wraps): A generic toolkit allows resources to be incorporated into a design infrastructure with minimal effort by the end user. Notice that the incorporation of a model within the toolkit allows for knowledge to be generated in context allowing a designer to interrogate knowledge for the who, what, where, when, and how the information was created.
- **Computational Backplane:** Components that are required for objects to operate in a distributed, homogeneous computing environment are included in a transparent, underlying infrastructure.

The characteristics of each component of the architecture will be outlined in the following sections.

7.1 Designer Activities

An implementation of the DSP Technique has been integrated into IMAGE and is adapted from earlier work done on a **D**esign **G**uidance **S**ystem (DGS).[15] This implementation compliments other process-related works in model creation and allows for DSPT Palettes to be generated (meta-design) and solved (process of designing).[21] An IMAGE root window with two Palettes visible is shown in Figure 15. The Palettes represent the highest level design processes for a High Speed Civil Transport Design. Also shown in the Figure is the main window for IMAGE. The window contains drag-and-drop facilities for data storage, printing, and help. The buttons located to the right in the window are used to (de)iconify Palettes and Design Specification browsers.

[Figure 15 goes here]

Combined with Palette activities, a Design Specification Editor is used to appropriate information activities. The Editor incorporates ideas discussed earlier, including schema evolution, fidelity support, automatic context generation and instance accumulation. An IMAGE root window depicting parts of this Editor is shown in Figure 16. The Form, Process, Model, and Heterarchy browsers are visible. In the browsers, design objects are created and associated with each other. A schema editor is also shown in the Figure. The schema editor allows for multiple schemas with their associated attributes to be defined. If this window were expanded, attribute instance values and their accumulation could be visualized. Figures Figure 15 and Figure 16 show the main user interfaces to IMAGE. The other sections of IMAGE described in sections to follow do not require user interfaces. Customizeable viewers for data and problem execution can be created to support those activities but are not required.

[Figure 16 goes here]

7.2 Available Assets

Available Assets are the second feature of the IMAGE infrastructure shown in Figure 14. These Assets, or resources, are the entities that are inevitably responsible for carrying out design methods. The Assets can be categorized as databases, visualization tools, optimization routines, geometric modelers, and algorithmic and heuristic simulation. The most familiar form of a resource is the computer program. A designer directs computer programs to calculate some desired information based on pre-determined algorithmic procedures. These programs may be combined in automated analysis modules that may incorporate heuristic controllers. Some examples of computer programs used by the aerospace industry include: ASTROS (a structural optimization code), FLOPS (an aircraft convergence code), ACSYNT (an aircraft convergence code), CONMIN (an optimization package), CATIA™ (a three-dimensional geometric modeling, simulation and analysis package), and ORACLE™ (a relational database).

7.3 Agent Collaboration

Agents are one of the key enabling technologies that bind IMAGE together, as shown in Figure 14. Agents allow for the meaningful creation of design knowledge by Available Assets (resources) during Design Activities. As discussed earlier, the information model used in the IMAGE implementation incorporates the use of Process elements. The instantiation of these elements as Agents results in knowledge generation in context.

The creation of an Agent requires three components: a Resource, a Model, and a Wrap. Resources are the Available Assets discussed in the previous section. Models have two components: the Process Model and the Implementation Model. The Process Model is the model incorporated into the process element. The model may be physical or intellectual. Models are typically based on mathematical formulations, engineering principles, or geometrical constructions.

The Process Model has typically been discarded or included only in external reference documents The use of Agents allows for Process Models to be explicitly defined. For example, a solids construction model used to represent complex solids in CATIA™ is shown in Figure 17. In words, the geometric process model describing the volume transformation would be:

> *In a volume transformation, an object is represented by an approximate solid computed directly from the exact volume. A volume is constructed from faces which, in turn, are defined by the edges that enclose simple or multiply connected regions of planar or complex surfaces.*

[Figure 17 goes here]

The Implementation Model, the second model component, captures the execution characteristics of the resource. Some of the items that are contained in the implementation model include: variable definition, file descriptions, units, execution characteristics, and platform dependencies.

A Wrap enables Agents to work in a collaborative environment. A Wrap is responsible for publishing models so that designers may employ the services of resource contained within, communicating information among resources while conforming to protocols and data exchange standards, and negotiating its services with other agents. In all a Wrap has six components: a Communications Interface, a Protocol Filter, a Model Interpreter, a Resource Interpreter, the (Graphical) User Interface, and a Low Level Compliance layer.

A generic scheme has been proposed that allows for Agents to be developed from existing available assets or new Agents to be developed that incorporate the functionality of new design environments and computing characteristics. References [22, 23] summarize generic Agent implementation schemes.

7.4 Computational Backplane

Design Activities utilize Available Assets in an accountable fashion through the collaborative use of Agents. The Computational Backplane allows for this process to occur using existing computer facilities, see Figure 14. The underlying architecture is a subset of what has come to be known as the National Information Infrastructure. The NII is simply a conglomeration of users and services on the Internet. Users range from end-users performing design functions to developers providing simulation services. Services range from information dissemination, such as on the WWW, to underlying transport mechanisms such as TCP/IP[5]. In addition, the NII also provides direction for new computing technologies such as fine-grained and parallel computing.

The wrap component of the Agent allows for collaborative efforts to occur through an intimate interface with the NII. Proper integration requires that the following services be standardized and made available: communications support, protocols, data representations, and ontologies. The NII incorporates rapidly expanding and evolving computing facilities. IMAGE has been designed so that new technologies may be incorporated and tested within the architecture without re-configuration.

---

5 Transmission Control Protocol / Internet Protocol

8. Future Directions

The computer architecture requirements for designing engineering systems have been outlined in this paper and addressed by a generic framework called DREAMS. In turn, this framework is being implemented on the computer as a design system called IMAGE. Two example cases are currently being implemented in IMAGE in order to assess the success of addressing the aforementioned design requirements. These two cases include a Simplified HSCT24E multidisciplinary design problem and a **G**eneral **A**viation **A**ircraft (GAA) open engineering system design.

8.1 HSCT24E

A simplified HSCT24E test case was implemented in FIDO. During its execution, a simplified HSCT was determined from a baseline model based on weight minimization and appropriate constraints. Execution was done on the distributed framework as controlled by an executive. This problem is being incorporated into IMAGE in two parts. First, the analysis resources in FIDO are being incorporated into IMAGE as agents. Second, the optimal problem is being reformulated as a Compromise Decision Support Problem.

8.2 GAA

The design of a family of GAA based on the 2, 4, and 6 seater aircraft configurations was performed using DSIDES directly linked to GASP, a General Aviation Synthesis Program.[24] A ranged set of top-level design specifications for the family of GAA was developed using a multi-objective Compromise Decision Support Problem in combination with Taguchi's robust design techniques and response surface models.[25] This problem is also being incorporated into the IMAGE framework in two parts. First, the response surface models and analysis resources from GASP are being incorporated into IMAGE as agents. Second, a common baseline model for the family of GAA is found by using DSIDES to find satisficing solutions for the GAA Compromise Decision Support Problem.

9. Acknowledgments

Funding for this article is provided by the NASA Graduate Student Researchers Program (NGT-51250) under the direction of NASA Langley's High Performance Computing and Communications Program. Related research is conducted under the New Approaches to HSCT Multidisciplinary Design and Optimization MDO contract (NGT-51102L). Software and hardware support is provided by the CAE/CAD Laboratory at the Georgia Institute of Technology. In addition, the collaborative efforts of the Georgia Tech Aerospace Systems Design Laboratory and the Systems Realization Laboratory are recognized.

10. References

1. Chapman, B., P. Mehrotra, J. V. Rosendale and H. Zima, "A Software Architecture for Multidisciplinary Applications: Integrating Task and Data Parallelism," Institute for Computer Applications in Science and Engineering, March 1994. NASA CR-194896, ICASE 94-18.
2. Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 1.2 ed., 1993.
3. Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck and V. Sunderam, *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*, Ed. Kowalik, J., Scientific and Engineering Computation, MIT Press, 1994.
4. Cutkosky, M. R., R. S. Engelmore, R. E. Fikes, M. R. Genesereth, T. R. Gruber, W. S. Mark, J. M. Tenenbaum and J. C. Weber, "PACT: An Experiment in Integrating Concurrent Engineering Systems," IEEE Computer, vol. 26, no. pp. 28-37, January, 1993.
5. Dovi, A. R., G. A. Wrenn, J.-F. M. Barthelemy, P. G. Coen and L. E. Hall, "Multidisciplinary Design Integration System for a Supersonic Transport Aircraft," Fourth AIAA / USAF / NASA / OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, September 21-23, 1992. AIAA-92-4841.
6. Finin, T., J. Weber, G. Wiederhold, M. Genesereth, R. Frtzon, D. McKay, J. McGuire, R. Pelavin, S. Shapiro and C. Beck, "Specification of the KQML Agent-Communication Language," The DARPA Knowledge Sharing Initiative External Interfaces Working Group, February, 1994.
7. Frank, G. A., J. B. Clary and B. L. Dove, "Design Automation for Concurrent Engineering," 1st AIAA National TQM Symposium, Denver, CO, November 1-3, 1989. AIAA-89-3207.
8. Hale, M. A. and J. I. Craig, "Use of Agents to Implement an Integrated Computing Environment," Computing in Aerospace 10, AIAA, San Antonio, TX, March 28-30, 1995. AIAA-95-1001.
9. Hale, M. A. and J. I. Craig, "Preliminary Development of Agent Technologies for a Design Integration Framework," Fifth AIAA / NASA / USAF / ISSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City, Florida, September 7-9, 1994. AIAA-94-4297.
10. Hughes, D., "Generic Command Center Speeds Systems Design," Aviation Week & Space Technology, vol. no. pp. 52-53, March 8, 1993.

11. Townsend, J. C., R. P. Weston and T. M. Eidson, "An Overview of the Framework for Interdisciplinary Design Optimization (FIDO) Project," NASA Langley Research Center, July, 1994. NASA TM 109058.
12. Bras, B. A. and F. Mistree, "Designing Design Processes in Decision-Based Concurrent Engineering," SAE Transactions Journal of Materials & Manufacturing, vol. 100, no. pp. 451-458, Warrendale, PA, SAE International, 1991.
13. Sobieszczanski-Sobieski, J., "A System Approach to Aircraft Optimization," NASA TM-104074, March 1991.
14. Sobieszczanski-Sobieski, J., "Multidisciplinary Design Optimization: An Emerging New Engineering Discipline," The World Congress on Optimal Design of Structural Systems, Rio de Janeiro, Brazil, August 2-6, 1993.
15. B.A. Bras, W.F. Smith and F. Mistree, *The Development of a Design Guidance System for the Early Stages of Design*, Ed. Oortmerssen, G. V., CFD and CAD in Ship Design, Elsevier Science Publishers B.V., Wageningen, The Netherlands, (pp. 221-231).
16. Stephens, E., "LEGEND: Laboratory Environment for the Generation, Evaluation, and Navigation of Design," Doctoral Dissertation, Georgia Institute of Technology, School of Aerospace Engineering, September 1993.
17. Rogers, J. L., "DeMAID - A Design Manager's Aide for Intelligent Decomposition User's Guide," NASA TM 101575, NASA Langley Research Center, March 1989.
18. Stettner, M., "Tiltrotor Multidisciplinary Optimization," Doctoral Dissertation, Georgia Institute of Technology, School of Aerospace Engineering, August 1995.
19. F. Mistree, O.F. Hughes and B.A. Bras, *The Compromise Decision Support Problem and the Adaptive Linear Programming Algorithm*, Ed. Kamat, M. P., Structural Optimization: Status and Promise, Washington, DC, (pp. 247-286), AIAA.
20. "Definiton of Requirements for an Aeronautics Affordable Systems Optimization Process," Proposal to NASA Langley Research Center, Madic Team #2, January, 1995.
21. Sharpe, J. E. E., "Integrated Platform for AI Support of Complex Design - Part 1: Rapid Development of Schemes from First Principles," Concurrent Engineering: Research annd Applications, vol. 3, no. 4, pp. 295-306, December, 1995.
22. Hale, M. A., J. I. Craig, F. Mistree and D. P. Schrage, "On the Development of a Computing Infrastructure that Facilitates IPPD from a Decision-Based Perspective," 1st AIAA Aircraft Engineering, Technology and Operations Congress, Los Angeles, CA, September 19-21, 1995. AIAA-95-3880.
23. Hale, M. A., "A Computing Infrastructure that Facilitates Integrated Product and Process Development from a Decision-Based Perspective," Thesis Proposal, Georgia Institute of Technology, School of Aerospace Engineering, January, 1995.

24. "GASP - General Aviation Synthesis Program," NASA CR-152303, Contract NAS 2-9352, NASA Ames Research Center, 1978.
25. Simpson, T. W., "Development of a Design Process for Realizing Open Engineering Systems," MS Thesis, Georgia Institute of Technology, School of Mechanical Engineering, 1995.
26. W.J. Fabrychy and B.S. Blanchard, *Life-Cycle Cost and Economic Analysis*, Ed. Fabrychy, W. J. and J. H. Mize, Prentice Hall International Series in Industrial and Systems Engineering, Prentice Hall, Englewood Cliffs, NJ, 1991.

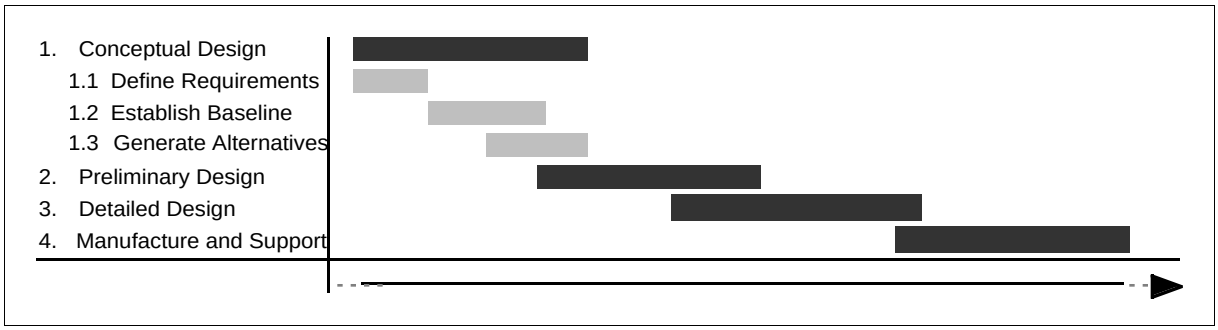


Figure 1. A Traditional Design Timeline

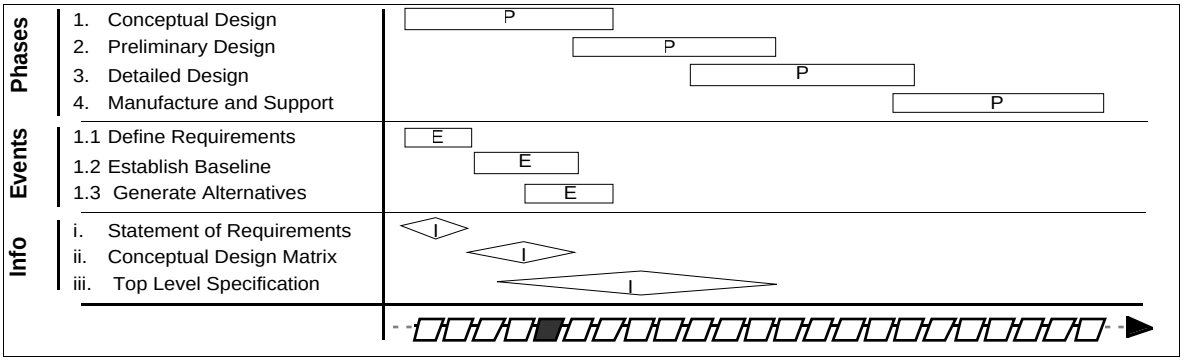


Figure 2. A Partitioned Timeline Corresponding to Figure 1

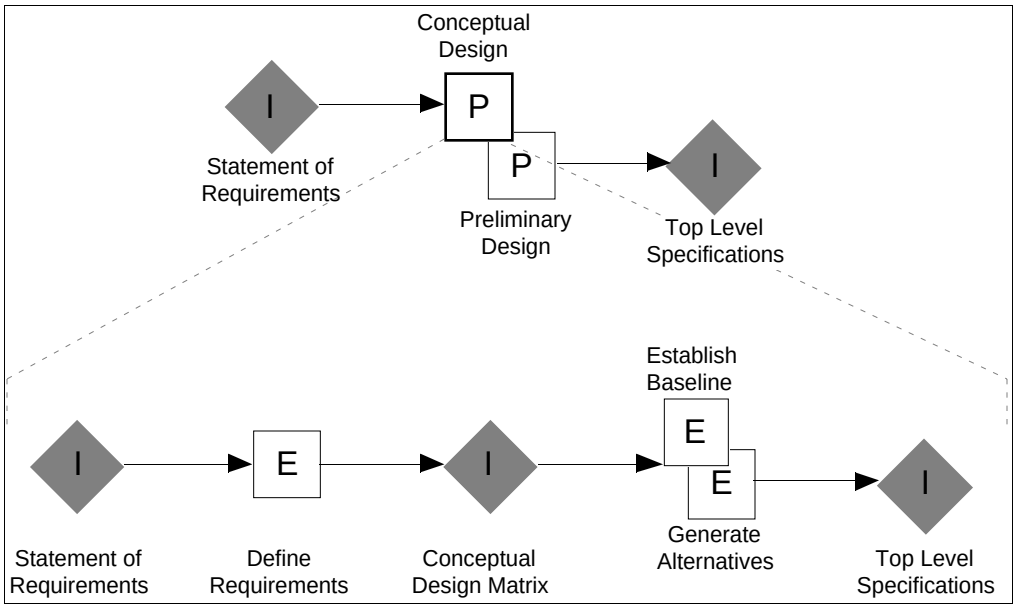


Figure 3. Support Problem Network Corresponding to Figure 2 (See Table 1)

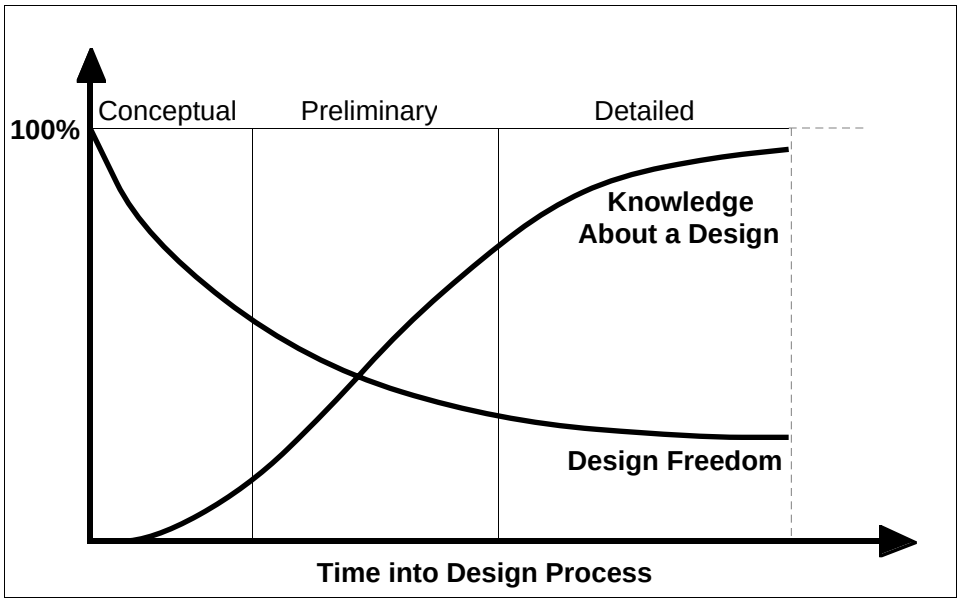


Figure 4. Design Freedom and Knowledge vs. Time [26]

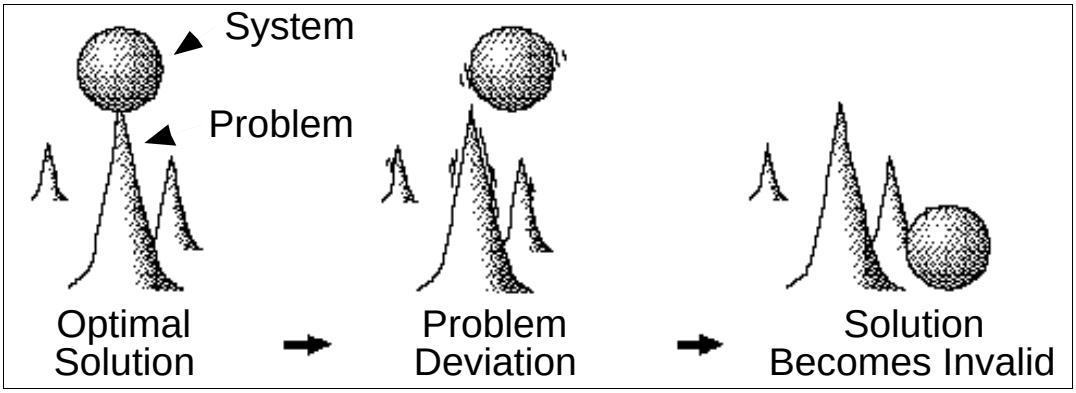
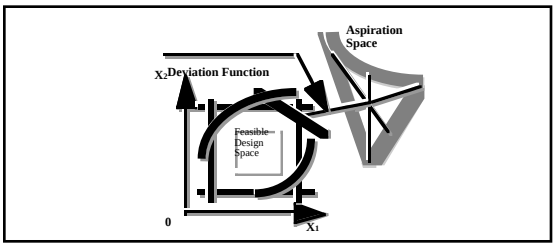


Figure 5. An Optimal Model

The Compromise DSP and Satisficing Solutions

In the Compromise DSP, the set of system constraints and bounds define the *feasible design space* and the sets of system goals define the *aspiration space*. For feasibility, system constraints and goals must be satisfied. A *satisficing* solution is that feasible point that achieves the system goals to the extent that is possible. The solution to this problem represents a tradeoff between that which is desired (as modeled by the aspiration space) and that which can be achieved (as modeled by the design space), see figure below. It is noted that the aspiration space (defined by the goals) is unlikely to overlap the feasible design space early in a design timeline. At some point on a design time-line, as a design matures, the aspiration space will overlap the feasible design space.

The mathematical form of a Compromise DSP is given to the right. Each goal has an achievement function, $A_i(\underline{X})$, a target value, G_i, and two associated deviation variables, d_{i^-} and d_{i^+} which indicate the deviation from the target. The range of values of these deviation variables depend on the goal itself. The product constraint $d_{i^+} \bullet d_{i^-} = 0$ ensures that at least one of the deviation variables for a particular goal will always be zero. The goals in a multidisciplinary system are not equally important to a decision-maker. To effect a solution, on the basis of preference, the goals may be rank ordered into priority levels. A solution which minimizes all unwanted deviations should be sought.



Given

An alternative to be improved through modification.
Assumptions used to model the domain of interest.
The system parameters

n	number of system variables
p+q	number of system constraints
p	equality constraints
q	inequality constraints
m	number of system goals
$g_i(\underline{X})$	system constraint function
$f_k(d_i)$	function of deviation variables to be minimized at priority level for Preemptive case

Find

Xi	i = 1, ..., n
d_{i^+}, d_{i^-}	i = 1, ..., m

Satisfy

System constraints (linear, nonlinear)

$g_i(\underline{X}) = 0$	i = 1, ..., p
$g_i(\underline{X}) \quad 0$	i = p+1, ..., p+q

System goals (linear, nonlinear)

$A_i(\underline{X}) + d_{i^-} - d_{i^+} = G_i$	i = 1, ..., m

Bounds

Ximin Xi Xi max	i = 1, ..., n
$d_{i^-}, d_{i^+} \quad 0$	i = 1, ..., m
$d_{i^-} \bullet d_{i^+} = 0$	i = 1, ..., m

Minimize

Case a: Preemptive (lexicographic minimum)

$Z = [\, f1(d_{i^-}, d_{i^+}), ..., fk(d_{i^-}, d_{i^+})]$

Case b: Archimedian

$Z = \Sigma\, Wi(d_{i^-} + d_{i^+});$ $\quad \Sigma Wi = 1;\ Wi \quad 0$

Adapted From

Lewis, K, and F. Mistree, "Designing Top-Level Aircraft Specifications: A Decision-Based Approach to a Multi-Objective, Highly Constrained Problem," 36th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamic, and Materials Conference ans AIAA/ ASME Adpative Structures Forum, New Orleans, LA, April 10-13, 1995. AIAA-95-1431-CP.

Figure 6. Compromise Decision Support Problem

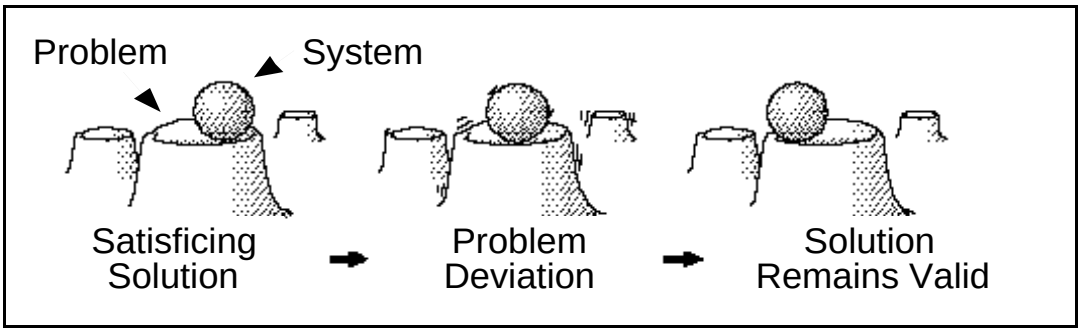


Figure 7. A Satisficing Model

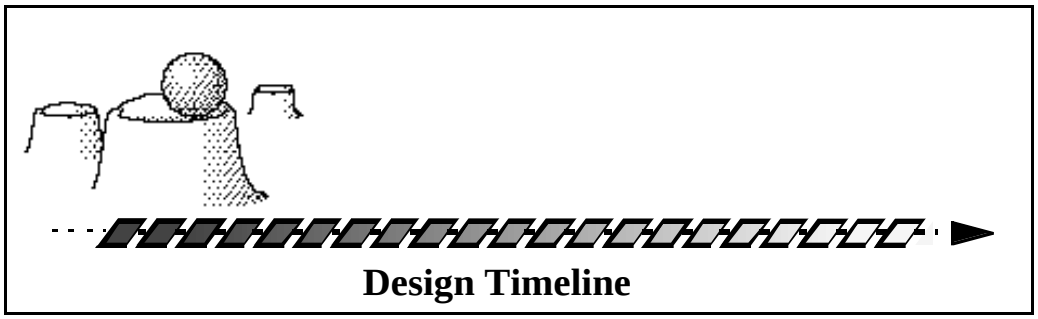


Figure 8. Satisficing Solutions Early in Design Processes

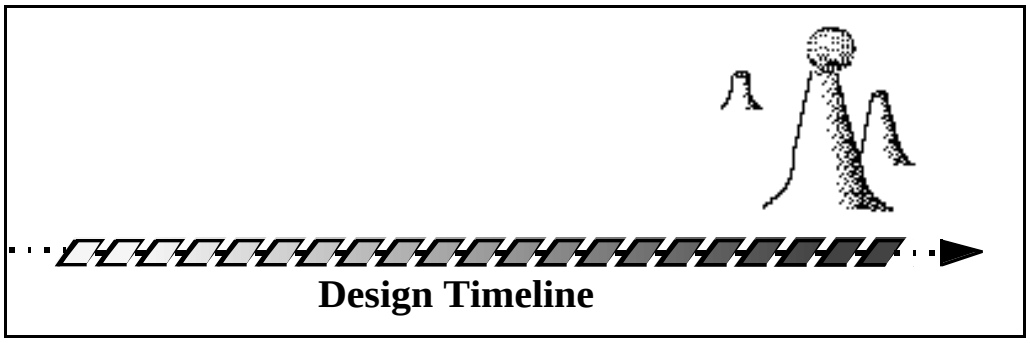
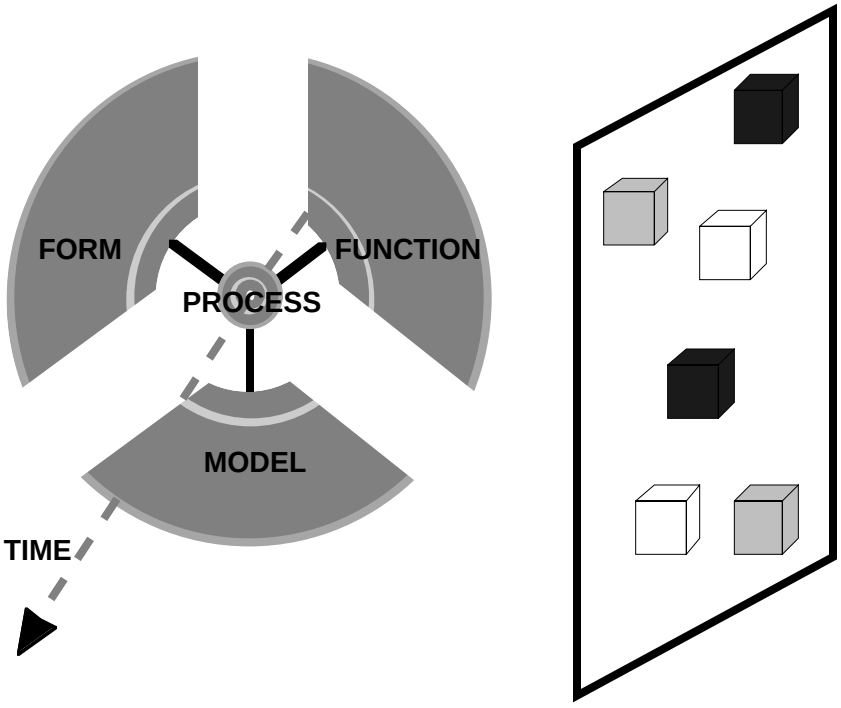


Figure 9. Optimal Solutions Later in Design Processes



a) Information Hierarchy

b) Information Heterarchy

Figure 10. Design Information

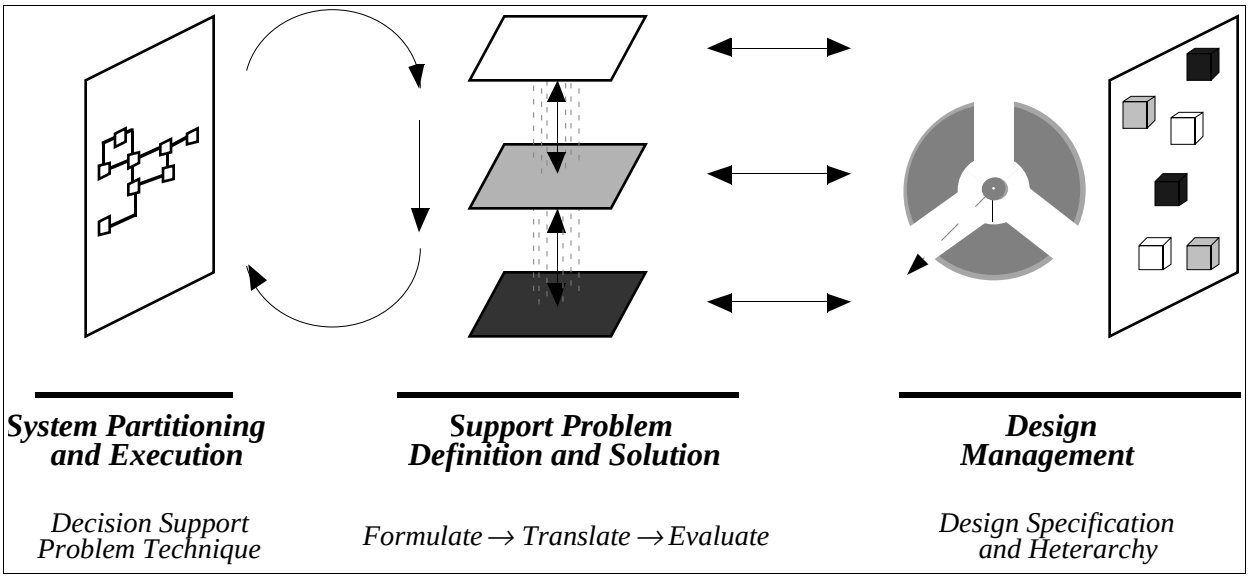


Figure 11. DREAMS Architecture for Supporting a Designer's Activities

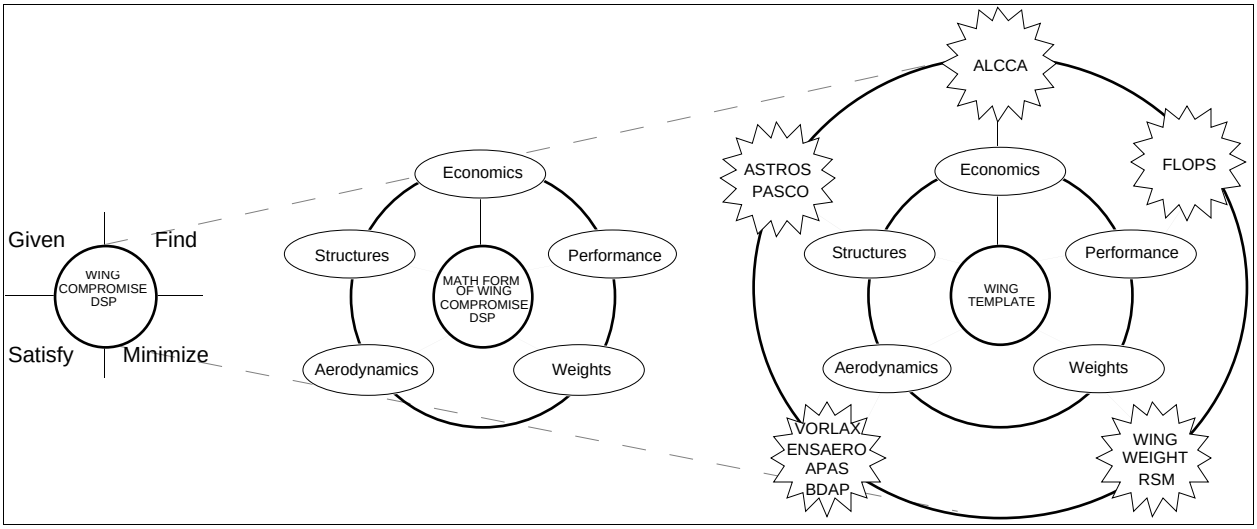


Figure 12. Multidisciplinary Wing Integration Compromise Decision Support Problem

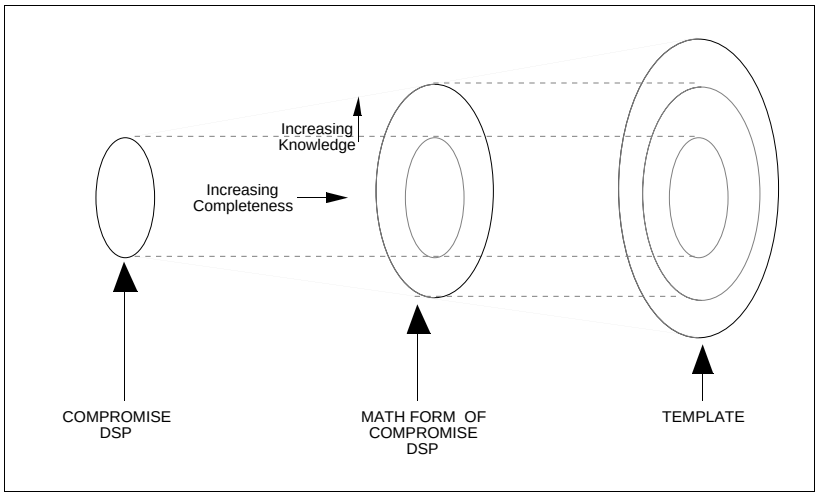


Figure 13. Support Problem Cone

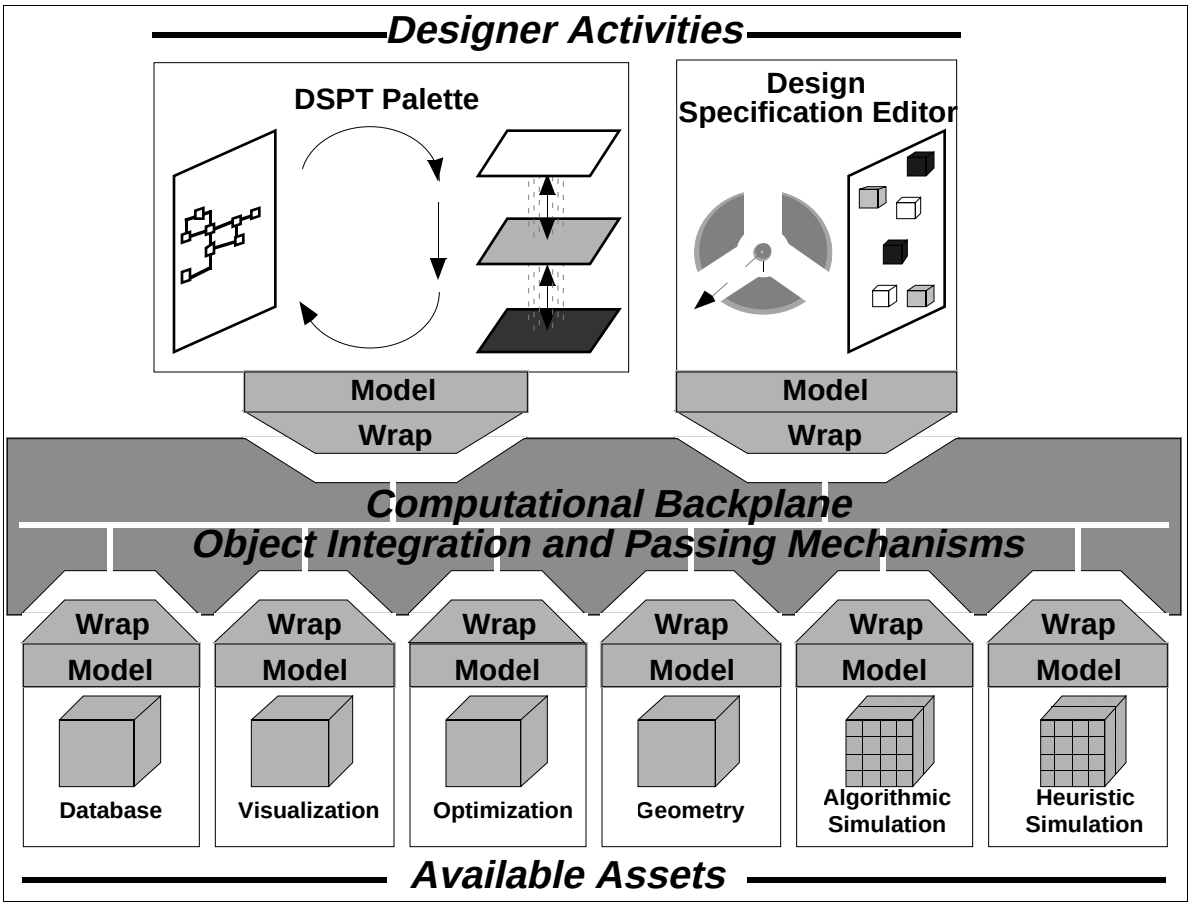


Figure 14. IMAGE Infrastructure

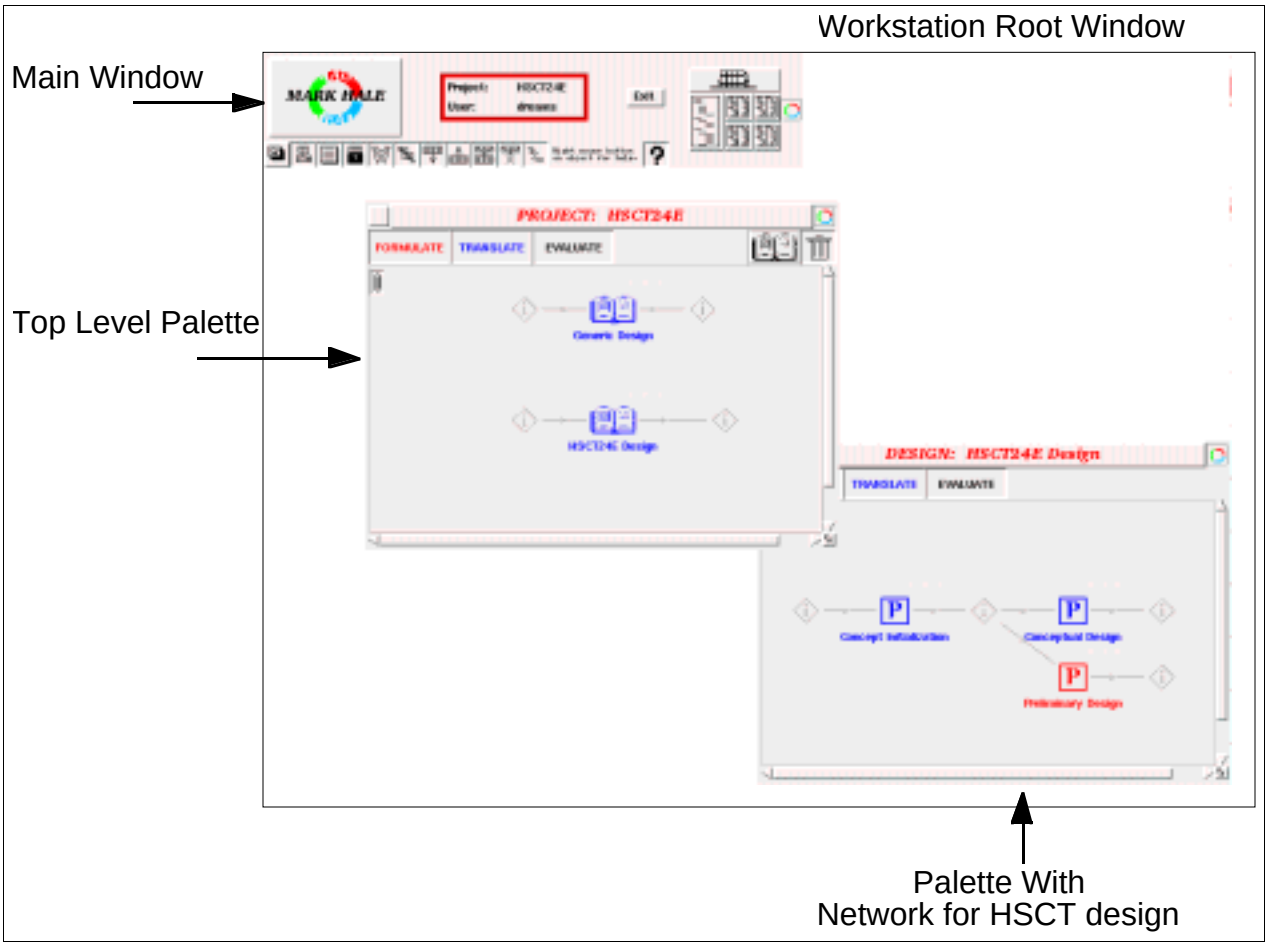


Figure 15. IMAGE Root Window With Palettes

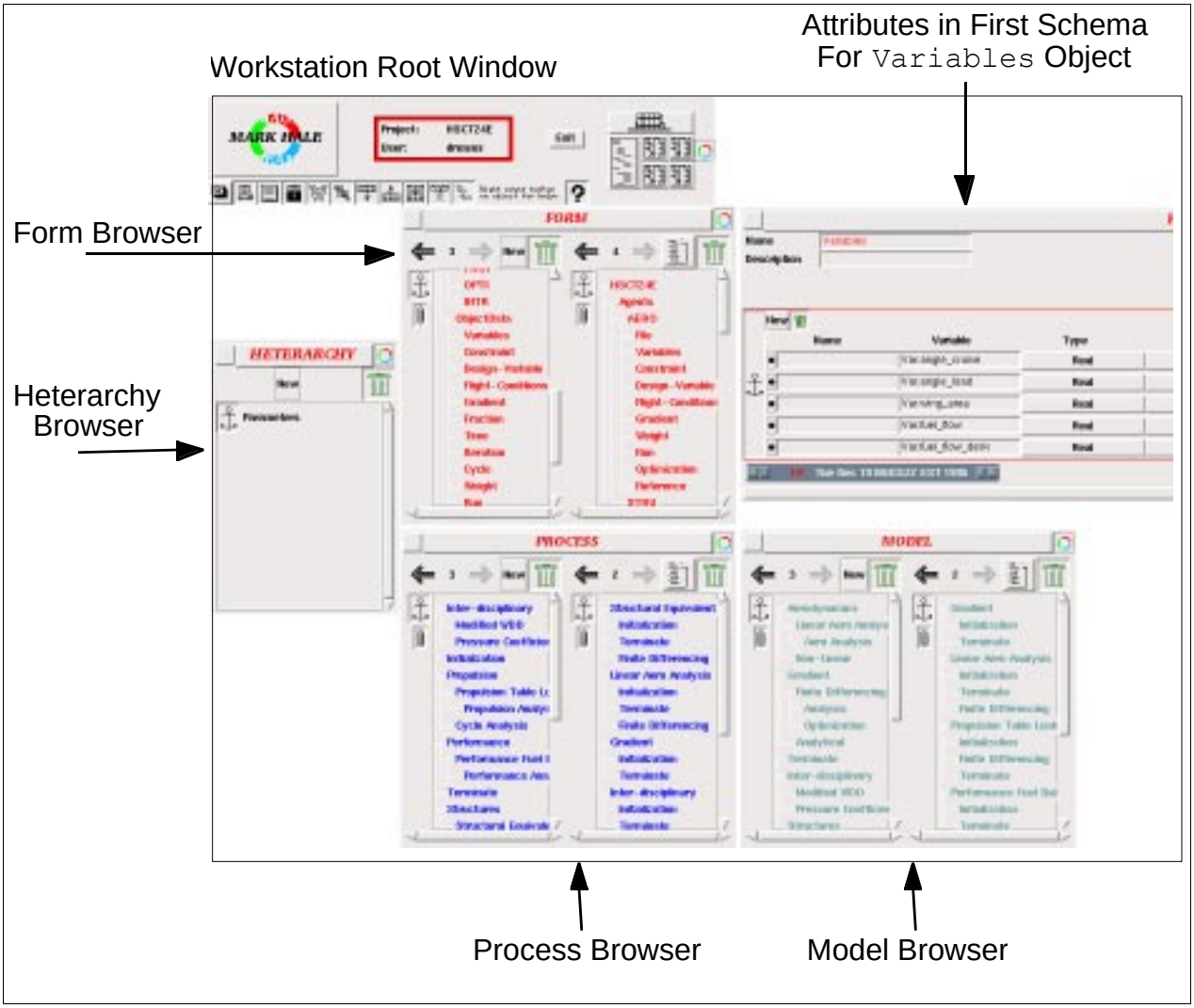


Figure 16. IMAGE Root Window With Browsers

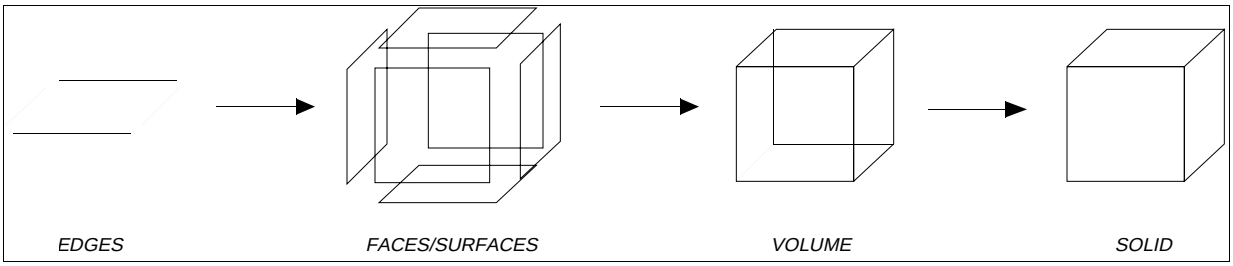


Figure 17. CATIA Solid Representation

Table 1. Network Icons

Support Problem	**Icon**	**Function**
Phase	P	Stages of development
Event	E	Coincidence of design activities
Task	T	Activity to be accomplished
Decision	?	Evaluation of a design based on its content
System	◎	Actual design components which may be real or abstract
Transmission Entity	**Icon**	**Function**
Information	i	Design information communicated between Support Problems

Table 2. Design Hierarchy Entities

Category	Function	Dimensions
Form	A mechanism by which a design can perform an activity	2
Function	An assigned activity a design is to perform	2
Process	The means by which a function is performed by a form	2
Model	An idealization of a process	2
Time	Either real or event-based	1
		$\Sigma = 9$

Table 3. Form Objects

Objects	Hierarchy
├Aircraft	├Aircraft
├Lifting Surface	├Centerbody
├Vertical Stabilizer	├Lifting Surface
├Horizontal Stabilizer	├Propulsion Unit
├Propulsion Unit	├Vertical Stabilizer
├Centerbody	├Propulsion Unit
	├Horizontal Stabilizer

Table 4. Increasing Fidelity for a `Lifting Surface`

├Lifting Surface
├Inboard Wing Box
├Spars
├Ribs
├Outboard Wing Box
├Spars
├Ribs

Table 5. Three Possible Schemas for Describing a `Lifting Surface`

Schema 1	**Schema 2**	**Schema 3**
Root Chord	Aspect Ratio	(X1,Y1)
Span	Span	(X2,Y2)
Taper Ratio	Taper Ratio	(X3,Y3)
Root t/c	Root t/c	(X4,Y4)
Tip t/c	Tip t/c	(X5,Y5)
Sweep	Sweep	Root t/c
Dihedral	Dihedral	Tip t/c
Twist	Twist	Dihedral
NACA Profile	NACA Profile	Twist
		NACA Profile