Abstract

The Automated Design Decision Support System is an ongoing Boeing Helicopters IR&D program to provide a concurrent engineering (CE) design environment for preliminary design. ADDSS research has focused on three components necessary for effective CE: a design representation for structuring and sharing design knowledge; design assessment using attributes as a design language; and an advanced computing architecture for the integration of design tools.

Concurrent Engineering: Design For the 'Ilties

Concurrent Engineering is a design philosophy that will improve design quality by integrating all the required disciplines into the design process, e.g. design for producibility or design for supportability. At Boeing Helicopters, design for supportability is a formidable challenge to today's rotorcraft design engineer. Both commercial and military customers have come to realize that the supportability of a system plays a role equal to performance, not only because of its effect on life cycle cost, but also because of its profound implications for fleet availability. All large airframe design organizations maintain a staff of advisors and/or evaluators (domain-specific engineers and experts) to address Reliability, Maintainability, Human Factors, Safety, Training, Design to Cost, and other supportability issues. However, the design engineer, being responsible for the final product, is ultimately in the best position to influence the supportability aspects of their systems.

Addressing supportability requirements early in the design process is recognized as an effective method of achieving a supportable end product. Supportability specialists have devised a number of methods to influence the preliminary design process, each with its own level of acceptance and success. Some of the tried methods include "board leaning" i.e. working alongside the designer, circulation of design guidelines and handbooks, and distribution of design for supportability checklists. Unfortunately, there are far too few specialists to address all the intricacies of a typical rotorcraft design, and this situation is further complicated by an overwhelming amount of supportability documentation, criteria, and guidelines.

CE Approaches

Design-Build Teams (DBT) and Product Development Teams represent one successful approach to practicing Concurrent Engineering. Design-Build-Teams are characterized as co-located, multi-discipline teams typically consisting of Engineering, Product Support and Manufacturing personnel, focused on addressing all aspects of a product early in the design process. This approach promotes open communication between organizations during preliminary design. DBT participants must be able to quickly evaluate technology alternatives in light of their disciplinary requirements and to draw accurately and rapidly on lessons learned and military specifications and guidelines. The low ratio between the number of engineering support specialists as compared to the size of the typical design engineering staff creates a fundamental problem. It is virtually impossible to adequately address all aspects of a complex system design in a timely manner with limited staff. As a result, DBTs tend to focus on top-level design concerns, leaving a multitude of design decisions solely to the responsible design engineer.

Recent studies of the design process have revealed other problems in the current approach to design team interaction. Participants have diverging ideas of the design process, a very large volume of data must be managed and shared among participants, and design guidelines and other important design-related data are difficult to retrieve. The process of engineering design has become so complex that the current design state is not easily characterized. An initial goal of Boeing Helicopters' Concurrent Engineering research and development effort was to develop the framework for a preliminary design environment that provided the design team with the following facilities:

- Fast and easy access to alternative components that satisfy certain functional requirements;
- Easy access to design guidelines from corporate, industry, or government sources;

29th ACM/IEEE Design Automation Conference®

Paper 32.2

506

0738-100X/92 $3.00 © 1992 IEEE
Designers explore, innovate, define and refine new design concepts, and search for solutions that are affected by numerous technical and non-technical constraints. During preliminary design, constraints change frequently. It is not possible to work out the details of each alternative solution. Instead, the design team must select alternatives by considering the information available during preliminary design. This leads to the designer's need for approximate methods to assess design solutions. Provisional design decisions evolve into guidelines, goals and commitments. Uncertain assumptions and the lack of complete information makes the designer's task of evaluating and coordinating preliminary design decisions very difficult. All the factors that influence and shape the design form the context in which decisions are made. Identifying and capturing interdependencies among design decisions are essential for concurrent engineering during preliminary design.

The goal of computer-based concurrent engineering is to ensure that all design decisions are based on the best collective knowledge available from all pertinent information sources, while recording and evaluating each significant design decision. This goal requires that all design information be extremely accessible, hence the special emphasis on design representation. The current lack of any standards for preliminary design information means that it is impossible to capture, share, review and build on design information. Concurrent engineering involves diverse groups of people. During the preliminary design process, these people need efficient methods for recording, communicating and sharing relevant information on complex and abstract design issues. By capturing design decision information, designers can take advantage of previous experience and make better design choices.

Automated Design Decision Support System (ADDSS)

Boeing Helicopters has established a Concurrent Engineering Lab which supports the diversities of a large engineering organization. The lab is structured around the concept of providing an integrated engineering environment to support multiple engineering domains as required for concurrent engineering. In order to support a computer based concurrent engineering environment, the research and development of computer systems must support the existing computing environment for all the design/build/support team members, as well as, the future facilities for these disciplines. The BH CE lab was designed with this philosophy in mind.

The CE Lab and the Advanced Computing Technologies group at Boeing have undertaken a multi-year Independent Research and Development (IR&D) program to research and identify methods to improve the Supportability/Operability of systems, equipment, or components through the timely influence of the preliminary design process. The Automated Design Decision Support System (ADDSS) and the Electronic Document Retrieval and Annotation System (EDRAS) are the products of multi-year research and development projects aimed at investigating computer-based Concurrent Engineering (CE). The objective of ADDSS is to create a computer-based environment for preliminary design that encourages and assists the design engineer to account for supportability, operability and producibility requirements early in the design process. ADDSS takes advantage of the flexible nature of preliminary design to add a structure that encourages the incorporation of Reliability, Maintainability, Safety, Human Factors, and Produicibility features without sacrificing traditional system performance goals. In ADDSS, the representation and modeling environment will permit designers to explore alternative formulations of the design task and to compare alternative solutions generated in terms of a given formulation. The interface between high-level concept models and their underlying computational tools(agents) enables a systematic analysis of evolving solutions. Feedback from these agents, such as design assessments, will lead designers to reformulate and generate alternative solutions until all design objectives are satisfied. ADDSS research has pursued the development of three components necessary in creating a powerful preliminary design environment: design representation for structuring and sharing design knowledge; design assessment using attributes, which form a common language for design; and an advanced computing architecture for integration of existing and future design tools. The ADDSS project is working on three facets of this problem, (1) the conceptual model of design information, (2) system integration issues and (3) the user interface, which are all critical for a successful implementation.

Design Representation

Extended Function Logic Model

In 1989, Carnegie-Mellon University-Engineering Design Research Center (CMU-EDRC), Pittsburgh, PA was contracted to research and identify a common language for preliminary design that is responsive to multiple engineering disciplines. CMU proposed the Extended Function Logic Model (EFLM) as the design representation language for ADDSS. Based on a Value Analysis and Engineering approach, this strategy focuses on identifying a product's functions rather than conventional hardware/software descriptions, implying what each element does or performs rather than what it is. This approach provides a common language for all specialists and allows the discovery of redundant activities at the early stages of the design. Further, the functional language motivates the generation of alternative design solutions that might accomplish the same thing with better quality or reduced costs. It is this design approach which most markedly affects
the preliminary design process and separates it from much of the current research work being done across industry and academia in design synthesis [2].

Figure 1 ADDSS Architecture

Boeing has translated EFLM theory into practice and created a design representation approach for ADDSS that is powerful yet flexible. Early user evaluations had indicated that not all designers were willing to work within the structure of the EFLM. Traditional approaches of creating system block diagrams without regard to functionality or with only mental images of the required functionality remain prevalent, hence the ADDSS design representation module has been designed to accept and operate effectively with a wide range of system representation styles. In addition, the ADDSS implementation takes into account that strict hierarchical representations of a system prove awkward at best and can often be incomplete or misleading. ADDSS offers sophisticated block linking mechanisms that capture the temporal or spatial relationships, including side effects, that may exist in a complex system design. The form of some of these linking mechanisms are the subject of continuing ADDSS research and development. The design representation allows a designer to build a system description that can be automatically or manually acted upon by a set of design decision support tools.

Object Representation

The ADDSS Design Representation will exploit object-oriented concepts for development and implementation. The object approach will enable designers to intutively structure knowledge as it is developed, while retaining the properties of that knowledge, and to model non-hierarchical, complex relationships between "atoms of knowledge". Object-oriented technology supports complex data types, relations, modularity and abstraction. Classes define the internal structural properties, model the behavior and declare the general type. This expresses levels of abstraction and provides the means for inheritance of class attributes and behavior. Individual objects are instances of a class and they encapsulate their own internal state information. An object-oriented concept, known as polymorphism, describes how different types of objects may respond similarly to the same message. Object-oriented technology is currently the best way to model the preliminary design information, given the need to assimilate large amounts of related textual, graphical, numerical, logical, structural and procedural information during preliminary design. Traditional methods, such as relational databases, just aren’t adequate for this purpose. The preliminary design information evolves from abstract concepts into very complex and interrelated design information.

The Workspace

The conceptual model of design information is based on an object-oriented schema. At the highest level of abstraction, we define a “workspace” that can represent literally anything. A workspace is a kind of container. Each workspace contains those objects needed to describe a particular concept. A workspace may contain other workspaces. This provides a way to describe and reference concepts at various levels of detail. The other types of ADDSS objects include elemental types and other container types. Detailed information exists in ADDSS elemental objects. The elemental types are used as building blocks to create more complex forms of design concepts. Elemental types include: text, images, numbers, rules, procedures and links. Container objects represent structural or organizational information. They aggregate or associate groups of objects, such as: sets, lists, arrays, networks and dictionaries. By combining ADDSS objects, the user can dynamically create composite objects to express virtually any kind of preliminary design information. For example, links can express point to point relationships (hypertext links). Sets can model groups of people in an organization, or attribute assignments. Lists show sequences of tasks, or prioritize relevant design information. Further research is needed to identify and define what extended types are needed to fully support concurrent engineering.

Through the ADDSS user interface, an empty workspace looks like a large drawing area (20K by 20K pixels). The user can draw blocks on the screen using a mouse and scroll a
window over this drawing area. These blocks are a visual representation of objects that the workspace contains. The user may create a new block in the workspace, indicate what type it represents (any elemental or container type), then proceed to specify more information about that block. Existing objects can simply be referenced using these blocks. For example, an engineer may create a functional block diagram of the major functions that a particular design must perform. The workspace that represents this design could include blocks for all the required functions and blocks that define how each of the functions are associated. There may be alternative ways to implement the functions. The user creates a workspace to represent each alternative. Each alternative carries with it certain attributes (see below), such as features, cost, material properties, fabrication (or procurement) information, and historical records. A workspace for each alternative could contain blocks representing design sketches, references to similar designs, performance data, industry standards and specifications, supportability assessments, and other analyses.

**Design Assessment and Attributes**

**Attributes**

ADDSS research has developed a concept simply called attributes. Attributes express relevant characteristics of design objects. Here, the term attribute does not equate to a "field", frame slot, or a column in a relational table. Attributes are not simply properties of a design object. From an object-oriented perspective, attributes are partial types (abstract classes) that encapsulate operations and state information to describe certain essential design characteristics. Attributes do not use multiple inheritance to bring characteristics into a class definition. Instead, attributes use inheritance only to build other attributes. Attributes are not inherited by non-attribute objects. ADDSS types, such as links and sets, describe the relationship between attributes and other design objects. An attribute is a kind of design object that is referenced by other design objects. This is a better way to specify a "has-attribute" relationship than using inheritance because attributes can describe characteristics which may be only temporarily associated with particular design objects. Therefore, this association may also include certainty factors (fuzzy sets). Attributes form the basis for common language in the design representation. Attributes capture the semantics of particular design concepts and discipline specific terminology. Attributes allow for indexing design objects. This attribute indexing mechanism allows interactive access, as well as procedural access, to retrieve relevant design objects. Furthermore, attributes and logical operators may be combined to form complex queries. Attributes also serve as indexes for other external systems that designers may access indirectly.

**Design Assessment**

The goal of design assessment is to provide automated means to flag potential problem areas early in the preliminary design process. Supportability assessment evaluates design alternatives with respect to supportability criteria. Using ADDSS, designers associate attributes with alternative solutions. The risks and benefits associated with each attribute are, in turn, associated with the alternatives. Supportability analysts and designers have defined attributes and relationships among attributes. They will be able to add more attributes as the design process progresses. Attributes associated with actual designs form the basis for assessments. ADDSS is able to assess designs based on metrics defined for specific attributes and certain combinations of attributes. This makes the rationale for design decisions more visible and trackable. If assumptions for a given design approach are either relaxed or further constrained, the ADDSS user can recheck the design using new attributes.

Design assessment is a form of decision making under uncertainty. Early design decisions may be based on uncertain information. Attribute associations between the rationale and the design option will model this relationship. ADDSS will apply appropriate mathematical techniques in the methods of attribute assignment under uncertainty. Rule-based knowledge stored in the design representation may also be applied to cross-check the preliminary design for compliance with specifications, interface requirements and ever-changing design goals. Again, attributes will be used to associate rule objects with design options so that inferencing techniques may be applied to the design assessment. Attributes allow designers to quickly retrace the design assessment for later review or correction. Furthermore, attributes are an integral part of the design representation.

**ADDSS Class Hierarchy**

ADDSS uses the OMTOS object-oriented database, from Ontos Inc. as the data management system for the capture, storage and retrieval of design information. The root of all OMTOS persistent objects is the "Object" class which has member functions for constructing, naming, storing, deleting and other database functions.

The root ADDSS class "A_Concept" inherits from the OMTOS "Object". Note, all ADDSS class names begin with "A_" to eliminate naming conflicts. The constructor for A_Concept makes three private objects: a reference to alias names, a timestamp, and a reference to the user. This class defines common functions that all ADDSS objects share. The accessor functions alias(), time(), and user() are available for any derived class. A_Concept also defines virtual functions (suffered operations) that each leaf class must implement, such as: show() and edit(). The show() function generally shows the user something; display a dialog box, open windows, or perform a procedure. The edit() function allows a user to modify the object's internal state. Other functions are available for ADDSS messages.

The abstract class A_Container is derived from A_Concept. It is a base class for all the ADDSS container classes: A_Workspace (ADDSS composite object), A_Collection (unordered set), A_Sequence (list/queue), A_Matrix (vector/array), and A_Network (association table).

A_Container includes the following public virtual functions:
virtual void member_names(SLIST& the_slist);
virtual Aggregate& container();
virtual Boolean is_member(A_Concept* the_concept);

The member_names() function returns the name of each object in the container, where SLIST is a convenient type for interfacing with XVT dialog listboxes (a user interface development toolkit from XVT Software Inc.). The container() function returns a reference to the internal ONTOS aggregate object. This function is particularly effective when used in conjunction with the ONTOS SQL aggregate query capability. The is_member() function searches for a particular concept either directly or indirectly referenced in a container.

ADDSS supports user definable messages that can be sent to a container. The container, in turn, sends the message on to each member. Nested containers propagate messages down to each leaf object. Each container must track the messages it sends so that circular references don't cause infinite loops. This is one reason why ADDSS messages are persistent objects.

The abstract and (grammatical) class A_Element is also derived from A_Concept. It is the base class for all the ADDSS elemental classes: A_File (text), A_Block (on a workspace), A_Image (graphics), A_Numeric (integer, real...), A_Rule (knowledge-base), and A_Procedure (ADDSS macro command language).

A_File class holds Unix files. Typically, these are text files that represent documents. Functions in this class can read, write, show and edit files.

A_Block is a class of visual objects associated with a workspace or a file. A block shows itself either inside a workspace or text file. In a workspace, a block is a rectangular box that can be moved around and linked with other blocks (shown as an arrow). In a text file, a block can highlight and scroll to a range of text given offsets from the beginning of the file. Attributes can be associated with particular blocks.

The A_Image class holds graphic images in GIF format files, with functions for converting from TIFF or PICT Macintosh file format. The show() function opens an X-Window and displays the image.

The A_Procedure class provides a macro command language capability with persistent storage. Procedural knowledge is stored in the ADDSS design representation as REXX language programs. REXX is a high-level interpreted language and SAA standard. REXX has two components: (1) a powerful interpreted macro command processor and (2) a set of C language interface routines. These tools permit ADDSS users to write application programs that dynamically link with ADDSS compiled (C++) routines. The REXX interface also allows external systems to interact closely with ADDSS object-oriented database.

A_Rule class encapsulates "knowledge-based" information. In ADDSS, the expert system provides inference techniques to process rule-based knowledge contained in the design representation. Design objects can contain logical information to be shared. Rules would be stored in the object-oriented database as elemental type objects. Each rule object will have an inference operation defined that tests its hypothesis by invoking backward chaining on an integrated expert system. The methods for implementing this are contained within the the rule class. Since rule objects encapsulate the methods for communicating with the expert system, the object-oriented database will automatically handle locking and serialization of the inference requests for the expert system. In this manner, the expert system may have multiple users, even though typical expert system shells are memory based (disk storage is only used to load the knowledge-base) and serve only one user at a time. It is also possible for multiple expert systems to service the ADDSS object-oriented database. ADDSS uses the NeXpert/Object expert system shell from Neuron Data Inc. This product has a C language callable interface that permits an embedded inference capability for the design representation. Rule-based knowledge stored in the design representation will be applied to cross-check the preliminary design for compliance with specifications, interface requirements and ever-changing design goals. Again, attributes will be used to associate rule objects with design options.

ADDSS Architecture

System Agency

Another facet of ADDSS research focuses on system integration issues. As a result, ADDSS employs an advanced computing architecture founded upon a System Agency. The System Agency is a new model for system integration built on the following concepts:

- **Client/Server architecture**: basic process control methodology
- **Network Computing**: parallel/distributed processing technology

ADDSS uses a multiple-client/multiple-server architecture to provide access to distributed heterogeneous data sources and applications. Client/server architecture here refers to a system in which the client performs local applications processing and user access, and the server retrieves data from the appropriate location, accesses the appropriate analysis tools as required, and returns the results to the client. Client programs request services of server programs. Server programs are started tasks waiting for requests from their clients. Multiple clients may request services from a single server. Conversely, multiple servers may be required to support a single client program. A client/server architecture provides the basic control for interactions among processes.

The System Agency extends the client/server model and implements network computing techniques. The function of the System Agency is somewhat analogous to a real-world agency (e.g., employment agency). An agency is a set of cooperative independent agents that serve as surrogates for clients. Instead of integrating each pair of N processes (or systems) with N*N-1 separate lines of communication, with only N lines the System Agency is responsible for making
connections, synchronizing client/server processes, broadcasting general messages and buffering real-time information. Parallel routines communicate with client and server processes. The System Agency is based on Network/Linda, TCP/IP, X/Open Transport Interface library and NFS shared files. ADDSS handles the coordination and communication between the software components and external systems through the System Agency.

**Figure 3 Agency View**

**User Interface**

The ADDSS user interface is accessible through Unix workstations and X-window terminals. This will be a prerequisite of the ADDSS operating environment. The user interface look and feel is consistent with OSF/Motif standards. The XVT library of C routines has helped build the user interface and supplements well the ONTOS library. XVT tools support user dialogs: scrolling lists, requesters, text display, windows, menus, fonts, mouse actions, keyboard handling, drawing, and printing. Lower level X-Lib routines are used to display bit-mapped images. The user interface is also linked with REXX and the X/Open Transport Interface libraries. The user interface interacts with the System Agency for asynchronous alerts and real-time communication. The user interface also includes a sensor process which informs and alerts the user as necessary. This server process responds well to asynchronous events in a concurrent engineering environment.

A non-interactive user interface through E-mail allows users forward copies of their relevant engineering design correspondence to the ADDSS system -- an important aspect of tracking the decision making process. Once the correspondence is in the system, it is then available to users through attribute assignments -- just like any other design information in the system.

**External Processes**

The external system interface permits other systems to work concurrently and interactively with the ADDSS system. ADDSS defines a Virtual User as an agent process that supports external client requests, as well as internal background processes. The Virtual User concept provides external systems with much of the functionality of the interactive user interface. External processes may also implement REXX procedures to "run" ADDSS procedures remotely. In addition, the ONTOS SQL interface allows ADDSS to act as a passive server for external clients. Using the POSIX compliant X/Open Transport Interface, Internet TCP/IP provides the network communications between ADDSS and external systems.

**Future Direction**

At Boeing Helicopters, ADDSS research will continue to investigate and demonstrate the mechanisms necessary for providing a foundation for a computer-based concurrent engineering environment. As these foundations are laid, the techniques, technologies and systems will be moved into production through incremental development of the ICE (Integrated Concurrent Engineering) system. The overall approach to implementing computer-based concurrent engineering at Boeing Helicopters will be a multi-phase effort to allow a growth process that will utilize existing tools, techniques, and methodologies and expand with the results of new techniques and technologies from research and standards. As the ADDSS research and other Boeing, university, and industry efforts produce useful CE techniques, they will be incorporated into the ICE project. A multi-phase approach will minimize the risk and ensure that the proven methodologies and technologies are incorporated into the ICE system as required.