Proceedings of DETC'02:
ASME 2002 Design Engineering Technical Conferences and Computers and Information in Engineering Conference
Montreal, Canada, September 29-October 2, 2002

DETC2002/CIE-34404

WEB-BASED DESIGN COORDINATION

Eric A. Domeshek
Stottler Henke Associates, Inc.
280 Broadway / 1st Fl
Arlington, MA 02474
(781) 643-1444
domeshek@shai.com

Elias Holman
Stottler Henke Associates, Inc.
280 Broadway / 1st Fl
Arlington, MA 02474
(781) 643-1444
holman@shai.com

ABSTRACT

This paper reports on a web-based groupware tool intended to support large distributed teams of engineers in modern engineering design practice. The Advanced Design Coordination Tool (ADCT) presents itself to members of a design team as a shared on-line set of engineers' notebooks with flexible editing, filing, viewing, browsing, and searching capabilities. Additionally, its use of product, process, and decision representations referencing an explicit domain ontology, and tied together by dependency links, introduces artificial intelligence technology that enables capabilities beyond simple unstructured design history capture. The resulting repository can capture a rich and structured *design rationale*, which, with the system's state management and versioning capability, enables recording active as well as rejected alternatives.

The system currently exploits the captured data to improve team coordination: it applies dependency processing algorithms to automatically generate notifications to appropriate team members, based on changes to design notes or the detection of design conflicts. Over time, the accumulation of structured design histories with rationales should provide the basis for a knowledge repository that can proactively offer advice on new design challenges. This paper first sketches the context that makes this tool desirable, and then describes the system's design and key capabilities. We finish with a discussion of limitations and future enhancements.

Keywords: Artificial Intelligence, Design Coordination, Design Rationale, Collaborative Work

WHAT CAN INFORMATION TECHNOLOGY (IT) DO FOR DESIGNERS?

Over the last decades, information technology (IT) has significantly changed the way most engineers do business. In that time, both IT and engineering itself have evolved considerably, so while we have reached a productive state in computer-based design support, it is by no means an end state, or even a steady state. This paper discusses a new sort of engineering support tool that directly addresses one of the major remaining sources of delay and error in engineering projects*: the need to coordinate the actions of large and often distributed engineering teams*. In addition to helping ease team coordination problems, this class of tool can also provide a solid base for proactive design knowledge management.

This opening section starts by characterizing some well-known important trends in general engineering practice, highlighting the demands they place on design support IT. It goes on to focus on several more specific, yet still relatively common, conditions under which a design coordination tool can be particularly useful. Finally we consider some constraints on plausible enterprise information systems for engineering, as a way of setting the stage for the following general discussion of our new tool.

General Trends

We consider here five interrelated trends in modern engineering practice: (1) increasing system complexity; (2) larger and more distributed design teams; (3) iterative design and refinement; (4) concurrent engineering; and (5) ever-changing requirements and circumstances. Together, these realities place demands on any enterprise information management system intended to support engineering design.

Increasing System Complexity

As technology advances our ambitions advance too. Engineers may have more powerful tools that make them more productive individually, but we ask more of them collectively. We have studied design processes in a range of domains, but aerospace offers some of the most graphic examples of the phenomenon. Consider, for instance, just two landmarks from the century-long history of aviation: the Wright Flyer (1903), and the Boeing 777 (1994). By nearly any metric you care to name—cost, number of parts, number of functions—the size and complexity of these two systems differ by orders of magnitude. It takes longer and costs more to design more complex systems, and more is at stake in the quality of their design. At some point, computerized record keeping becomes necessary just to track what is going on in a design project. Design support IT needs to be scalable, potentially to support thousands of engineers working for a decade or more (and preserving their results many decades or longer).

Larger & More Distributed Design Teams

As suggested, despite improvements in individual productivity, larger design projects require larger design teams. The need to tap a widening variety of ever more specialized expertise leads to design teams that include members from many locations and even organizations. Escalating costs and risks likewise drive organizations into teaming arrangements, and lead to distributed design teams. The Boeing 777 project involved up to 10,000 engineers spread across Boeing and hundreds of subcontractors located all around the world. As a consequence, design support IT needs not only to be hugely scalable, but also to work well in a distributed environment.

Iterative Design & Refinement

Design methodologies come and go, but it seems safe to say that for any but the most routine problems, the days of straightforward "waterfall" design are over. Today, most complex designs are developed through a succession of versions, where each iteration incorporates more detail, informed by decisions and analytic results produced during earlier passes. IT for design needs to support some notion of versioning for data in order to support successive refinement. Ideally it should support not just revisions, but also multiple simultaneous explorations, as well as merging of results from different tracks.

Concurrent Engineering

Along with iterative design comes concurrent design, in which representatives of all phases of a product's life-cycle, and as many relevant design specialists as possible, contribute their perspectives to the design process early on. This approach to design tends to expand the circle of design team members, and puts extra stress on communication and organizational supports. Again, design support IT must support ever larger and more distributed teams, and must provide ever tighter coupling between the work all those team members are doing.

Changing Requirements & Circumstances

No matter how clear a project's goals seem at the start, change is a fact of life. Iterative and concurrent design can help promote effective risk management by enabling exploration of difficult aspects of the design early on. Still, such explorations are only of value if you are willing to adapt to the results—that is, if you are prepared to shift your approach, or even reframe your problem. Even if your goals and core solution are not subject to significant changes, the world around you is. Things change so quickly now that within the life-span of a large-scale design project, it is not unusual for some key technology or market condition to shift, enabling or requiring significant rework. Coping with such changes becomes exponentially harder as the size and distribution of the project team increases. Change is likely to remain an essential element of the design universe. An IT infrastructure that directly addresses the team coordination problem can reasonably be expected to produce a high payoff.

Special Conditions

In addition to the general trends just discussed, there are several other common, though not universal, characteristics of design environments that can place significant additional demands on design support IT infrastructure. Here we consider four: (1) a need for extensive formal documentation; (2) a need to support incremental refinement of fielded systems; (3) design of evolving product families with long life spans; and (4) design oriented towards configuration of semi-customized products on a shared underlying architecture, using common subsystems.

Need for Extensive Formal Documentation

Generally speaking, large projects cannot be run on individual memory and good intentions. When one organization is paying millions or billions of dollars to another to get some job done, the managers and their lawyers, at least, are going to insist on clear requirements and some indication of how the ultimate artifact meets those requirements. Value-based accounting adds additional pressure to document work products and their relationship to a work-plan that addresses the agreed-upon requirements. Left to their own devices, many engineers would skimp on documentation, in part because there are more interesting and immediately productive things to do with their time, and in part because they suspect that what they are doing may soon need to be revised in any case. In addition to storing and preserving documentation, design support IT should do its best to make the authoring of coherent, comprehensive documentation easy.

Incremental Refinement of Fielded Systems

For a significant class of complex, long-lived systems, the initial fielding of a product is not the end of design and development. Military systems, for instance often see several rounds of field upgrades during their effective life. When a team seeks to augment or modify an existing artifact, it does well to understand the current state of the system in all its relevant detail. In the likely event that much of the original design team is unavailable, new work must proceed on the basis of documentation and reverse engineering supplemented by limited advice from those old team members who may be available. Significant time can be saved, and mistake avoided if there is a good record, not only of what decisions were made in the original design, but also of *why* they were made (and of what alternatives were explicitly considered and why they were rejected). In this context, design support IT systems that capture, preserve and provide access to design history and rationale are especially valuable.

Evolving Product Families with Long Life Spans

The circumstance of evolving product families is a variant of the field upgrade situation just considered. The arguments in favor of exploiting design history and rationale are similar, though here, in what is potentially a fresh design, an even wider range of decisions and alternatives may be relevant than in the case of a mere upgrade. The question of where to hew to original work and where to diverge is more open, and therefore potentially benefits from even more insight into the original design process that can inform those decisions. Also, records of several earlier design efforts may be relevant, with pieces from a number of different family members potentially entering into the design of the latest variant.

Configuration from Common Subsystems & Architectures

Our final specialized condition is again similar to the previous two, but repetitive design from common components places a different emphasis on the historic knowledge an IT system should be managing. In this case, there are likely to be experiences with analyzing (or even fielding) particular combinations of components that, based on past experience, can be predicted to work well or poorly together. Decisions on such combinations in the context of new designs should be informed by past experience with such combinations. With this class of design problems, IT facilities initially conceived as documentation management, and that through extensions of content and access might reasonably be termed *knowledge management*, plausibly takes a final step towards case-based *design assistance*.

Constraints from the Real World

The design support tool we aim to introduce is essentially a distributed team-oriented tool—in modern IT parlance, an instance of *groupware*. To complete the picture of both the demands and constraints on such design support software, we survey a set of facts of modern IT life: (1) groupware has to earn acceptance in any organization; (2) Web browsers have become the universal front-end, especially for distributed systems; (3) Relational Database Management Systems (RDBMSs) have become the universal back-end, especially for any system that manages volumes of valuable enterprise data.

Groupware Has to Earn Acceptance

Groupware is either a tremendous IT success story, or an ongoing series of failures, depending on how you define the category. Email and newsgroups have clearly become ubiquitous. More proprietary applications such as Lotus Notes are successful mergers of such generic capabilities with additional shared data (mostly document) management, and more recently web accessibility. But application-specific groupware has had more than its share of failures. Easy ways to fail with groupware include asking too much of users (in learning curve or ongoing effort), and offering too little reward in return for the effort (Grudin, 1994). Any design-support tools aimed at coordinating teams of engineers must provide users and easy point of entry, and must provide fairly immediate rewards to the front-line designers (and not just to their managers).

Web Browsers as Universal Front End

Nowadays most practicing engineers—indeed most design team members whatever their background or function—are somewhat computer-savvy. In most environments, they have easy (if not continuous) access to networked workstations. Still, every specialty has its own special software tools. What most computer users share is familiarity with an operating system, facility with a set of basic office applications, and comfort with the web and web browsers. These browsers are, to some extent, becoming the new universal front end for information systems. Adherence to an evolving set of standards affords reasonable power to control displays and interactions, while allowing unprecedented portability across end-user platforms. Browsers now routinely bridge across workstation operating systems. They are starting to bridge across entirely different classes of devices (e.g. PDAs, tablets, cell-phones, and even set-top boxes). When a system is naturally about distributed information management, the most commonly accepted way to get the data to the end user, or to support basic data entry and interaction, is through a web browser.

RDBMSs as Universal Back End

Just as the web has sewn up the IT front end, RDBMSs have a lock on the back end, and not just for traditional structured data. Object databases are really still just an interesting niche technology, while RDBMSs are being extended to inter-operate more cleanly with object-oriented programs and their data. Files and directory structures may accumulate large amounts of work in process, but RDBMS vendors are now aiming to provide a more secure version of file systems. The core “vault” capability that managers value in PDM systems is essentially subsumed by industrial strength RDBMSs’ ability to manage the contents of arbitrary files along with their metadata. It seems clear that if you want to earn the right to manage important enterprise data, then you have to store it in an acceptable industry standard RDBMS.

SOLUTION FRAMEWORK

This section gives a high-level overview of the tool we have developed covering first the primary view—or metaphor—that shapes the end-user experience, secondly the major sorts of functionality provided by the tool, and finally, some notes on the technology used in developing the tool.

Metaphor

We started out wanting to attack the problem of confusion, wasted effort, and error generated by poor coordination among the members of design teams. The only approach we could see to the problem involved capturing and managing a relatively detailed network of information about the progress of a design project. A major question, then, was how to get access to the information our system needed?

From interviews with practicing aerospace engineers, we learned that the front-line mechanism for capturing design information is the "engineer's notebook." Every engineer we spoke with agreed there was such a thing as an engineer's notebook, and readily accepted the need to keep notes. However, even within a single company, engineers could not agree on the exact form or content of an engineer's notebook. For many it was a physical bound paper notebook. For others, it was a loose-leaf binder. In some projects, it was a rack of binders kept in a central location. For others it was an on-line directory structure of files. In a few cases, projects maintained some kind of on-line database for their notes.

Each of these diverse solutions had problems and limitations. Paper-based systems generally do not easily support effective indexed access, can only be in one place at a time, and are prone to ending up nowhere at all (when physical notebooks are misplaced, or their owners retire). Computer-based systems may or may not provide adequate search and browsing capabilities, and the information is often opaque to the computer. Higher end systems tend to be harder to learn and to use. For instance, we heard major complaints about usability from front-line engineers during the introduction of PDM systems in their organization.

Based on this input, and mindful of the need for groupware to ease and earn its way into actual use, we chose to structure our system around the metaphor of a shared library of on-line engineers' notebooks. There is an easy entry route into using the system: log into the system through your web browser, click to create or open a notebook, then click and type to add new notes to your book. Each note is simply a block of text, optionally accompanied by attached files that might contain graphics, supporting data, and so on. The system supports rich search and browsing capabilities ranging from paging through notebooks chronologically, to running attribute (e.g. author and date) and text based searches within and across notebooks. The system also provides security between projects, and privacy between users.

Functionality

Making it easy to put generic textual notes on-line is just a start. Our system also provides a useful taxonomy of note types that reflect an analysis of design products, processes, and decision-making; links among these typed notes can capture the ways in which some aspects of an evolving design depend on other facts and decisions. More ambitious users, then, can start creating and interlinking notes of these more specialized types, and begin to exploit the more advanced features of the system. This section introduces some of those features at a high level.

Capture, Organize, and Safeguard Design History & Rationale

Our engineers' notebooks are intended to promote capture not just of design history, but also of design rationale—the system provides notes to structure the arguments and decisions behind assertions. When you want to come back later and change something about a design, it is a major advantage to know *why* things were originally done in a particular way—what other aspects of the design a feature depends on. When you want to transfer aspects of a design from one context to another, it is crucial to understand the dependencies on the original context. Specialized note types and inter-note linking mechanisms enable structured representation of history and rationale. A web-based interface and database-backed storage ease capture, and ensure reliable but restricted access to information about past designs.

Encourage Formal Recording/Tracking of Decisions & Dependencies

Our way of capturing decisions relies on an interrelated set of specialized note types that together allow users to characterize a structured decision process in detail. We encourage users to frame *issues*, lay out alternate *options* for resolving such issues, as well as sets of *criteria* for evaluating the available options, and then to *evaluate* each option on each criterion. This aspect of the system was inspired by Rittel's notion of an Issue Based Information System (IBIS) (Kunz and Rittel, 1970; Conklin and Begeman, 1988; Ballinger, Banares-Alcantara, and King, 1993).

We further encourage users to link assertions about the design product and process either as *reasons* why issues were framed and options chosen, or as *consequences* of selecting particular options. Finally, we allow many assertions to be expressed as simple logical formuli refering to a domain-specific language of engineering concepts. The result is essentially a parallel structure: a set of human-readable notes describing decisions with their reasons and consequences on the one hand, and a formal dependency structure linking semi-formal assertions on the other hand.

Exploit Dependency Structures to Proactively Manage Team Coordination

Given a formal dependency structure, the system can make computations about the possible effects of changes and conflicts, and generate targeted notifications to appropriate team members based on those computations (delivering such notifications in the same on-line web-based environment where the users manage their notebooks, notes, and related discussions). When a support for some decision changes, the decision itself may need to be revisited; when a decision is changed, the assertions it supports may need to be revisited. Whoever is responsible for the decisions or assertions can be notified of the relevant changes. Links to data files potentially allow the system to trigger revision cascades based on changes to such files (which might, for instance, represent analysis results that help justify a decision). Formalization of design assertions potentially eases detection of design conflicts, and allows exploitation of the dependency network to find those engineers implicated in creation of the conflict, and thus suggest an initial set of team members to be drawn into discussions of how to resolve the conflict.

Support Structured Knowledge Management within and Across Projects

Given both a formal decision dependency structure and formalized assertions, it should become possible to provide useful access to pieces of past projects based on similarity in the requirements, prior commitments, critical issues, and options. This is an area we have only just begun to explore, so while the promise is clear, we count it as future work. The final section discusses this idea in somewhat more detail.

Technology

In this section we offer some brief notes about the technology underlying our system. We touch first on the gross system architecture, second on the rationale underlying our choice of tools and languages, and finally the origins of the most distinctive data structures and algorithms in the system.

3-Tier Web-Based System

Our system has been designed as a three-tier system built on commodity and COTS components, and open (web-based) standards. For the client tier we support standard COTS web browsers—in particular, MS Internet Explorer 5.x and higher, and Netscape Navigator 6.x and higher. The browser is treated as a sort of simple universal GUI framework. The back end is an industry standard SQL relational database; in the current system we use Oracle 8i, but ports to other SQL databases are quite possible and under active consideration. The database is responsible for enforcing data security, and supporting ad-hoc reporting. To the greatest extent possible we have tried to make our database schemas relatively conventional, so the system can inter-operate with legacy systems, and so that data is not trapped in peculiar formats that only our system can understand. The middle tier is our own application server, built as a set of extensions to a standard web server. It contains the custom logic for generating data views (e.g. libraries, notebooks, notes, notifications, and discussions) and editing layouts, as well as the core algorithms for change propagation and notification generation. However, to allow for easy server replication, it minimizes caching of data, allowing distributed users to, for the most part, see up-to-date data at the level of quite fine-grained database commits (e.g. individual note creation and edits).

Java + Standards-Compliance for Portability

For portability the application server is built in pure Java using servlets running in an open-source Java web server (currently Jetty 3.1); it should run on any platform that supports Java 1.3 or higher. For broad compatibility and easy access from client systems, the system sends only standard HTML with CSS and some light JavaScript to the browser. Likewise, on the database side, we have so far strictly limited the amount of non-standard code in the system, preferring to see how far we can get with standard SQL. Two major exceptions to date include exploitation of Oracle's indexed text retrieval options, and hierarchical queries. Nonetheless, the existence of a basic database abstraction layer and the clear factoring of these exceptions make us optimistic about the level of effort required to port to an alternate database. A port to PostgreSQL is under consideration since it would enable delivery of a fully functioning portable version of our system with a clean installation and no dependencies on costly third-party software packages.

AI-Based Representations and Dependency Networks

The final, and most unconventional, aspect of the technology underlying our system is the exploitation of techniques from artificial intelligence (AI) to represent and process information about design projects. Our work derives from earlier work by Petrie (1993) on the Redux design architecture, which in turn was based on AI research on truth maintenance systems (see Forbus and deKleer, 1993 for a comprehensive survey). Our note taxonomy encompasses a straightforward basic representation system covering design processes, products, and decision-making; in addition, we support definition of arbitrarily fine-grained domain-specific (installation-wide) ontologies to enable detailed formalization of design assertions. It is the introduction of these AI-based techniques that allow us to make the leap from on-line design notebook to active design coordination tool. The next section provides considerably more detail about the current implementation of our ideas, focusing on design representation, notification generation, but including discussion of other core system features such as discussion groups and versioning.

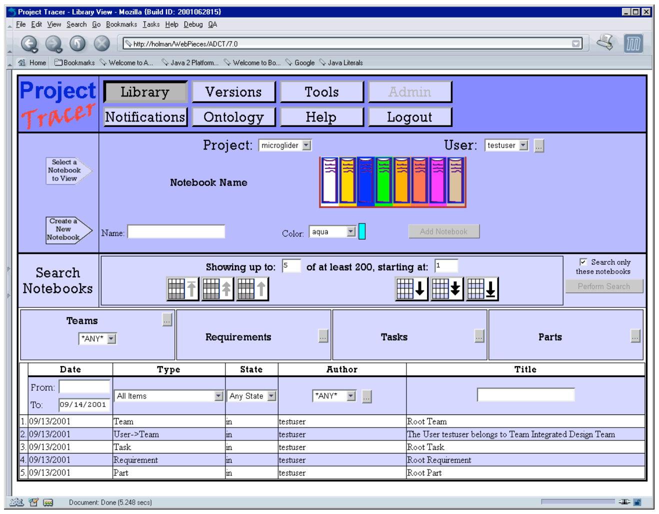
THE ADVANCED DESIGN COORDINATION TOOL (ADCT)

The current instantiation of the ideas outlined so far is the Advanced Design Coordination Tool (ADCT). ADCT was developed for NASA, as an exploration of how to address the kinds of design support IT issues sketched in the introductory section. During the system's development, we worked especially closely with engineers from Raytheon's Knowledge Center Southwest located with their Missile Systems Division. The examples in this section reflect the current state of ADCT (v3.2.3), and our access to data on a particular Raytheon design exercise (the unmanned, unpowered, but guided "microglider" surveillance platform).

Notebooks On Line

We introduce ADCT by presenting a collection of screen shots that together illustrate the major system modes that carry the notebook metaphor. Figure 1 shows the Main Window from ADCT (sometimes also known as "Project Tracer") in *Library View*. The browser-hosted screen presents a bookshelf full of notebooks for a chosen project and user. As the mouse moves over each book, the book's name appears next to the shelf. Clicking on a notebook opens a new window for viewing the chosen book. When the user is viewing their own notebooks, a type-in field, color menu, and button below the bookshelf allow for creation of new notebooks in the current project. Currently, it is entirely up to the users to decide how they want to organize their information into a collection of notebooks. The sample data from the "microglider" project shown here is partitioned into notebooks based on the major types of data handled by the system (e.g. Teams, Tasks, Parts, Requirements, Notes, Assertions, and Issues).

The bottom half of Figure 1 shows a "Table of Contents" (TOC) for the entire bookshelf—essentially a searchable index to the system's contents. Each notebook, when opened, has its own TOC page as well for more focused searches. When search criteria are entered (e.g. into the blanks at the head of each column of the result-set table), the result set is filtered down to a more restricted set. At any time, clicking on a row of that table jumps to the notebook page containing the particular note represented by that row.


Project Tracer - Library View - Mozilla {Build ID: 2001062815}
File Edit View Search Go Bookmarks Tasks Help Debug QA
http://holman/WebPieces/ADCT/7.0
Home Bookmarks Welcome to A... Java 2 Platform... Welcome to Bo... Google Java Literals
Project Tracer
Library Versions Tools Admin
Notifications Ontology Help Logout
Select a Notebook to View
Create a New Notebook
Add Notebook
Search Notebooks
Perform Search
Teams
Requirements
Tasks
Parts
Document: Done (5.248 secs)

Project: microglider User: testuser

Notebook Name

Name: Color: aqua

Showing up to: 5 of at least 200, starting at: 1

☑ Search only these notebooks

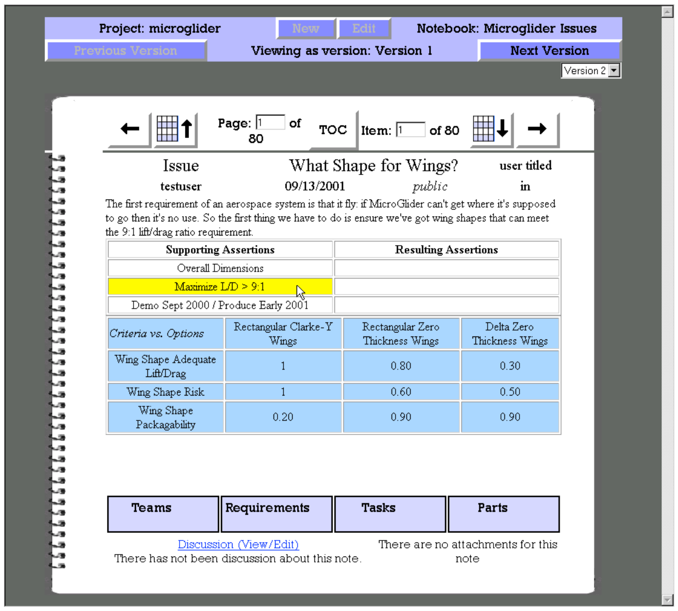
Teams: *ANY*

	Date	Type	State	Author	Title
	From: To: 09/14/2001	All Items	Any State	*ANY*	
1.	09/13/2001	Team	in	testuser	Root Team
2.	09/13/2001	User->Team	in	testuser	The User testuser belongs to Team Integrated Design Team
3.	09/13/2001	Task	in	testuser	Root Task
4.	09/13/2001	Requirement	in	testuser	Root Requirement
5.	09/13/2001	Part	in	testuser	Root Part

Figure 1. ADCT's Main Window (Library View)

Figure 2 shows a sample page from the Issues notebook. Such notebook pages are the primary way of viewing detailed information in ADCT. The idea is that each entry in the system becomes a page in some notebook. Each notebook page tells you what project you are working in, what notebook you are looking at, what version you are seeing, and what page/item you are viewing within the notebook. When viewing a note page, not only can you create a new note, but you can also edit the current note.

The top margin of each notebook page is taken up with indicators and controls for moving through the notebook (e.g. what page number is being viewed, which item from the currently selected set). The bottom margin is taken up with displays and controls for labeling and annotating the current note (e.g. attaching data files, or threaded discussion entries). The middle of the page contains the note proper: first a set of attributes (e.g. type, name, author, data, state), then a block of free text, and finally—for all but the most basic generic notes—some tabular layouts of structured data and relationships to other notes.



Project: microglider Notebook: Microglider Issues

Viewing as version: Version 1

Page: 1 of 80 Item: 1 of 80

Issue What Shape for Wings? user titled
testuser 09/13/2001 *public* in

The first requirement of an aerospace system is that it fly: if MicroGlider can't get where it's supposed to go then it's no use. So the first thing we have to do is ensure we've got wing shapes that can meet the 9:1 lift/drag ratio requirement.

Supporting Assertions	Resulting Assertions
Overall Dimensions	
Maximize L/D > 9:1	
Demo Sept 2000 / Produce Early 2001	

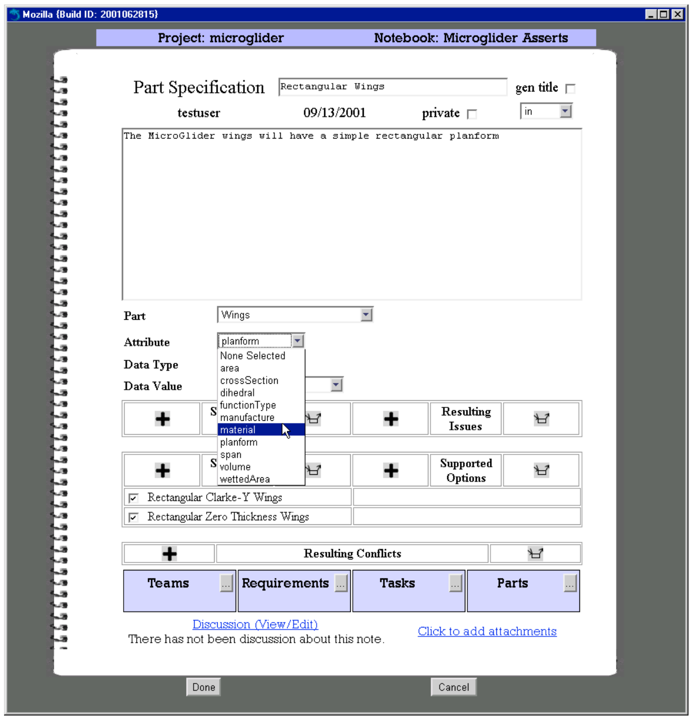
Criteria vs. Options	Rectangular Clarke-Y Wings	Rectangular Zero Thickness Wings	Delta Zero Thickness Wings
Wing Shape Adequate Lift/Drag	1	0.80	0.30
Wing Shape Risk	1	0.60	0.50
Wing Shape Packagability	0.20	0.90	0.90

There has not been discussion about this note.

There are no attachments for this note

Figure 2. ADCT's Notebook Window Viewing a Sample Issue as an Issues Notebook Page

The note shown in Figure 2 is hardly a simple generic note. Issue notes—with their central role in ADCT's model of a structured decision process—are actually among the most complex data structures in the system. Figure 3 shows another of ADCT's more complex note types—a Part Specification—this time in its editable form. Note also the pop-up menu of wing attributes, which illustrates the system's exploitation of an ontology defining basic domain concepts. In the following subsections, we discuss ADCT's typology of notes in some detail.



Project: microglider Notebook: Microglider Asserts

Part Specification Rectangular Wings gen title ☐
testuser 09/13/2001 private ☐ in

The MicroGlider wings will have a simple rectangular planform

Part: Wings
Attribute: planform
Data Type
Data Value

☑ Rectangular Clarke-Y Wings
☑ Rectangular Zero Thickness Wings

There has not been discussion about this note.

Figure 3. ADCT's Notebook Window Editing a Sample Part Specification (Pop-Up Menu with Attributes of a Wing)

Typed Notes

As suggested earlier, ADCT notes fall into two major classes: product/process notes record *what* was considered or decided, while rationale notes record *why* certain decisions were reached.

Product/Process Notes Record "What"

Product/process notes record possibilities about the design and the design process. Specialized note types provide ways to capture requirements, parts breakdowns, part specifications, team structures, tasking assignments, and other information. ADCT currently recognizes fourteen common categories of product/process note, and future versions will be open to installation-wide customization. Figure 4 shows relationships among several different types of notes:

In return for picking an appropriate note type, users get some direct benefits. One advantage of putting types on notes is that it becomes easier to find any particular piece of information you might be looking for, as the system can restrict searches based on note type—e.g. only look for requirements. Another advantage of typed notes is that they can carry specialized information. For instance the kind of note that records the assignment of a task to a team can store references to the team and the task.

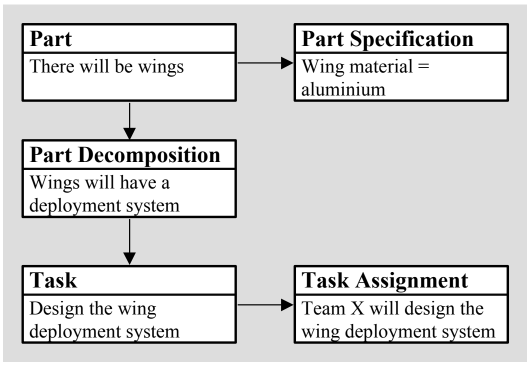

Part
There will be wings
Part Specification
Wing material = aluminium
Part Decomposition
Wings will have a deployment system
Task
Design the wing deployment system
Task Assignment
Team X will design the wing deployment system

Figure 4. Sample Product/Process Notes

Those data references, in turn, serve several purposes. First, they allow the system to recognize the *meaning* of the note: if the task is ever assigned to that team again, the system can recognize the duplication; if the task is assigned to a different team, the system can recognize the conflict. Second, they provide another way for users to browse and access information: when looking at a note describing a team, it is easy to find the team's tasks. When looking at a note describing a task, it is easy to find the team to which a task has been assigned. Finally, since ADCT's relational data store makes it relatively easy to generate ad-hoc reports such as lists of tasks that have not yet been assigned to any team.

Rationale Notes Record "Why"

Rationale notes let designers record the reasoning behind the commitments captured in product or process notes. Several kinds of rationale notes work together to capture a structured decision process:

Issue Notes describe decision points—either major ones that might be the focus of entire trade studies, or minor ones that are resolved with a little thought by a single engineer.

Option Notes describe alternate possible resolutions of Issues. A major Issue might have several well-analyzed Options. A minor Issue might start out with only a single recorded Option, but the Issue provides a place to attach new Options, should the original Option not pan out.

Criteria Notes describe how the Options for an Issue are to be evaluated. Criteria usually derive from Requirements. The number of Criteria is likely to vary with the importance of the Issue.

Evaluation Notes fit in an Issue's Option/Criteria grid (see Figure 6). Each Evaluation records a discussion and a heuristic rating of how a given Option fares with respect to a given Criterion.

Decision Notes provide a place to summarize the reasons for choosing a particular Option for a particular Issue; the Decision serves to mark the Option as active. If designers do not want to frame an entire Issue with Options, Criteria, and Evaluations, ADCT provides a view that focuses on Decisions and does not require managing the other structures behind it.

Conflict Notes exist to record problems caused by mutually incompatible assertions. When a Conflict is noted, Decisions that support the offending product/process notes must be revisited.

Consider a set of rationale notes from the microglider design. Raytheon's unmanned, guided glider was supposed to stow in and deploy from a cylindrical canister. With requirements for low cost, solid reliability, and a high lift/drag ration, wing deployment became one of the major design issues. Designers came up with eight possibilities for the wing deployment, including wings that telescoped out, pivoted forward, pivoted backward, or fanned out. Criteria included packagability, cost, reliability, weight, and space. Figure 5 shows a simplified version of the decision and rationale to have wings pivot forward. Figure 6 shows this Issue's Options, Criteria, and Evaluations in a grid layout.

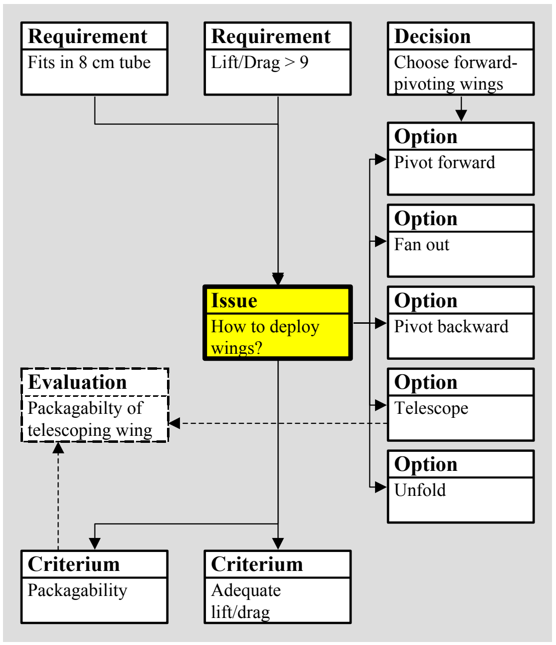
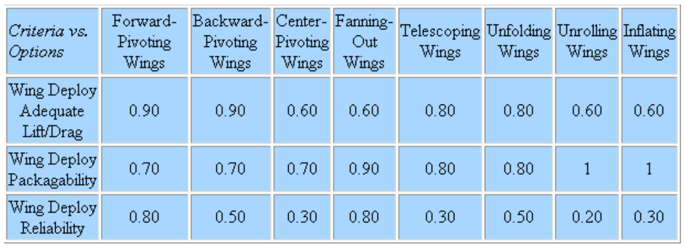


Figure 5. Rationale Notes for Wing Deployment Issue



Criteria vs. Options	Forward-Pivoting Wings	Backward-Pivoting Wings	Center-Pivoting Wings	Fanning-Out Wings	Telescoping Wings	Unfolding Wings	Unrolling Wings	Inflating Wings
Wing Deploy Adequate Lift/Drag	0.90	0.90	0.60	0.60	0.80	0.80	0.60	0.60
Wing Deploy Packagability	0.70	0.70	0.70	0.90	0.80	0.80	1	1
Wing Deploy Reliability	0.80	0.50	0.30	0.80	0.30	0.50	0.20	0.30

Figure 6. Rationale Grid for Wing Deployment Issue

Linking "What" & "Why"

Product/process notes capture "what" may possibly be true about the design or design process, while rationale notes captures "why" certain decisions were reached. To complete this picture, these two kinds of notes need to be linked together. The linked structure is called a *dependency network*. Key rationale items—notably Issues and Conflicts—depend on prior product/process assertions, and such assertions, in turn, can depend on rationales—most especially Options.

Conjunctions of product/process notes can raise design Issues. In the microglider example, Requirements for packaging and reliability combine to raise the Issue of wing deployment. Each Option has one or more product/process assertions that result if the Option is chosen. The "forward-pivoting wings" Option supports notes that introduce pivots and position them with respect to the wings and body.

These dependency links can be used to maintain consistency as the repository contents change. Product/process and rationale notes can be either *active* or *inactive*. To preserve a complete record of the design process, ADCT does not delete notes; it marks them as inactive. If *any* of the product/process notes leading to an Issue become inactive, that Issue may no longer be relevant and might also need to become inactive. If an Issue becomes inactive, its Options should become inactive as well. If an Option becomes inactive, the product/process notes it leads to may deserve to be inactive as well. But since aspects of product and process design can be supported by more than one Option, the rule here is that *all* the supports for such a note must become inactive before it is reasonable to make the note itself inactive.

A Conflict, like an Issue, exists because of some combination of facts about the project. In this case, the facts don't simply present a challenge to be solved. Instead, they represent an inconsistency to be resolved by removing some subset of the conflicting facts. Dependency links to Conflicts and the optional links to Options and from Issues play into the activity calculations as well. Figure 7 shows how incompatible decisions for two design issues can lead to a design conflict. Specifically, the Decision to deploy wings by fan out implies a delta-shaped wing which conflicts with a previous Decision to use a rectangular wing planform. A Conflict note identifies the two conflicting Part Specification notes.

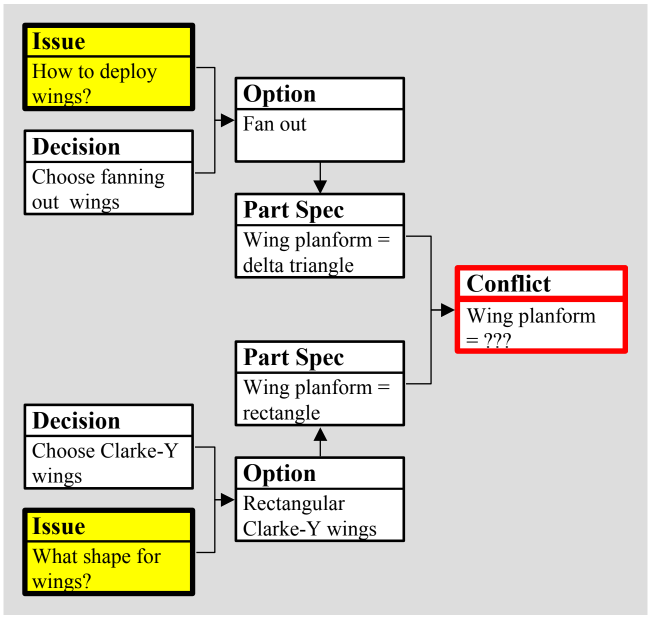


Figure 7. Incompatible Decisions Leads to Conflict

Designers can create, edit, link, and visualize notes using browser displays for each note type. When a note has links to other notes, the titles of the referenced notes appear in the display, and are clickable, allowing easy browsing to connected notes. In this way, the structure of ADCT's network of notes is of direct value to designers trying to learn about the status and history of a project.

Design Change Notification

Automatic change notifications are the other major payoff of a well-organized, fully linked set of design notebooks. Using dependency information, ADCT can automatically route change notifications to appropriate team members. Pending notifications trigger a flashing icon in the user's display. From their notifications page users can review pending notifications, and from a notification, they can easily jump to a relevant Note. So when are notifications generated?

The rules using dependency links to determine active/inactive status give a feel for the mechanism. When any team member rescinds a Decision, the corresponding Option becomes inactive. This may mean that some of the Option's dependent product/process notes no longer have any active support. In that case, the authors of those notes are notified and encouraged to make them inactive. If a product/process note is removed from active status, any Issues or Options it supports may need to be revisited, so their authors can also be notified.

ADCT also lets users turn such active/inactive decisions over to the system on an item-by-item basis. In that case, the system will not stop to notify an author that the status of an item might need to be changed; it will go ahead and change the status itself, then notify the author that the change has been made. In these situations, the effects of changes can ripple forward through the dependency network.

When a Conflict is noted, the system identifies relevant parties by searching backward through the network. Each of the product/process notes implicated in the Conflict is traced to the active Options that are its supports, and the authors of those Options' Decisions are notified of the Conflict. Those decision-makers are invited to join an ad-hoc discussion group about the Conflict. Conflicts will normally be resolved when one of the decision-makers rescinds one of the Decisions.

By exploiting dependencies, ADCT greatly improves project coordination. Its notification mechanism ensures that all the right people find out, in a timely manner, about changes and conflicts that impact their work. Its discussion group mechanism provides automated support for asynchronous and distributed resolution of conflicts. In combination with the versioning mechanisms discussed below, dependency processing enables "what-if" explorations of alternate designs when changes are necessary.

Discussion of Designs and Changes

Conflict discussion groups are just one example of ADCT's general facility for managing threaded discussion attached to Notes. If a Note is public, team members are free to view it and to post comments in a public forum. ADCT stores the history of such discussions along with the basic Note.

Conflict discussions are special: they initiate automatically, subscribing a selected group of users who are then notified of new postings. Conflict discussions also allow inclusion of other users into the group: either authors of other Decisions further upstream, or managers of those already in the discussion. Escalation of a Conflict discussion is useful when the original notified group cannot settle on a resolution.

Versioning for Concurrent Design

As noted in the introduction, the waterfall design method is frequently inappropriate, and design teams need the freedom to explore alternatives and iteratively refine all aspects of the design, as additional information is generated. To support iterative and concurrent design, ADCT supports a branching model for storing multiple versions of all designs. Any user can split a branch off a previous version. Users can designate any version as their current working version, and so long as that version is unlocked they can perform edits that register in that version. Each version only records its differences from prior versions. In a future release, we plan to add support for exporting work from a branch back to a main-line version.

FUTURES

ADCT 3.x is currently available as a customizable product. Under a project sponsored by NIST, we are developing a set of extensions to ADCT that address systems integration tasks, and emphasize development of formal enterprise models (ontologies). The Domain Expert Collaborative Ontology Development Environment (DECODE) we are building for NIST provides an opportunity to stretch and refine aspects of ADCT. Still, promising extensions to ADCT far outstrip available resources. Here we discuss some of the more important usability and capability enhancements that we hope to pursue in the future.

Increased Integration with Engineering Infrastructure

ADCT should access project information with as little user effort as possible. Most engineering shops have existing processes and legacy systems that generate and manage information potentially of use in ADCT. For instance, planning information about task breakdowns, time windows, team structures, and resource assignments are commonly managed in tools like MS-Project. As part of our original NASA research, we built a link to import MS-Project data into ADCT. Bi-directional links, and links to other third-party tools (e.g. requirements management tools, analytic tools, etc) are both possible and desirable.

A major limitation of the current implementation is that references to files "owned" by other applications occur only as attachments to ADCT notes. Such attachments are uploaded to the ADCT server and effectively lose their connection to the original file. Without a process running on each user's machine, it is hard to check for changes in the original files—changes that ought to prompt notification for designers to revisit decisions based on such files' contents. One clean general way to deal with this problem, especially in modern engineering environments, is to forge links with a PDM system that is already responsible for tracking changes to, and centralizing storage of, design-related files.

Finer-Grained Access Controls & Distributed Storage

ADCT currently provides access controls at the level of entire projects. It uses standard database mechanisms to enforce such controls, so they cannot be circumvented, even by going around the ADCT application server. The system also provides note visibility control (private/public) for individual notes. This is enforced only by the ADCT application server, and so can be circumvented. Future versions of ADCT should provide more fine-grained access controls and should enforce them all at the database level. This can be achieved, for instance, by greater reliance on database stored procedures, and doing so would likely also improve system performance. The down-side is increased implementation complexity and decreased portability (which is equivalent to still more implementation complexity). For applications in large organizations (and especially "virtual" organizations) this may be a necessary tradeoff.

Virtual organizations—project-specific teams of otherwise separate organizations—are becoming a more common, and present special difficulties with regard to design information management. The general issue is: How much information will companies in limited cooperation (and frequent competition) willingly share with one another? From this question spring others, such as: If design information is going to reside in a database, who's machine will it live on, and who is going to have access for what purposes? In ADCT, design notes link to other notes so that dependency processing can produce targeted notifications. Can we support linking and dependency propagation if project databases fracture along organizational boundaries, and access varies with business policies and contractual requirements?

Engineering Unbound: Tablets and PDAs

Yet another form of decentralization we must be prepared to support is the proliferation of small portable devices with intermittent network connections in the hands of an increasingly mobile workforce. Engineers have never spent all their time chained to their desks and workstations. Along with the limitations of common workstation I/O devices, the need that most engineers have to get out and around has constituted a major argument for paper-based over computer-based engineering notebooks. In the coming few years, the widespread adoption of tablet computers and PDAs will begin to remove these final barriers—if the software is ready to migrate to the new platforms, and if the problems introduced by this form of distribution can be successfully overcome. In truth, products like the Psion netBook and netPad, with their embedded Java VM and available Oracle replicated data server are already reasonably positioned to meet these challenges. Unfortunately, the price/performance ratio of such solutions is likely to keep a mobile version of ADCT out of the hands of the average engineer for a few more years.

Application-Specific GUIs & I/O

ADCT provides a fairly generic and homogeneous view of notebooks and their contents—a note is a note (though displays of specific note types are augmented with their type-specific data and links). The Issue note display comes the closest to offering a truly custom layout that gives guidance about how the note type is supposed to be used (specifically by displaying a grid of options versus criteria with room for evaluations, and highlighting of current and past decisions). ADCT was built to support "generic" design, and consciously shied away from adopting, embodying, or enforcing any particular design methodology. While a good first-pass strategy for a general tool, it is not necessarily the place to stop.

In our work for NIST on DECODE, methodology is front and center. The ADCT note types will (largely) be the same, but the user interface is being tuned to a particular view of how systems integration and ontology development projects should proceed. For instance, instead of generic Part notes, DECODE will provide a custom interface for introducing systems, subsystems, and inter-system information flows.

Easier Formalization & More Formal Reasoning

DECODE, with its focus on ontology development, is also going to take some initial steps towards making it easier to introduce more formalism into notes describing the commitments entailed by design decisions. For instance, requirement and part-specification notes can contain the equivalent of logical assertions about the attributes and relationships of parts, but only if a sufficient ontology is on hand to form the assertions. But in truth, engineers are only going to bother with such formalization if the system can repay their efforts in some obvious way. This is an area that will require work beyond the scope of DECODE, but we can identify a number of ways in which formalized notes can be useful: detecting and avoiding duplication of content; detecting and reporting conflicting assertions; and automatically interoperating with formal analysis tools.

Proactive Case Retrieval

One of the more powerful applications of the materials accumulated in ADCT—and one that can be improved by the availability of formal assertions about captured designs—is using the corpus of past designs as a mine for reusable design fragments and lessons learned. Such an extension would provide the kind of knowledge management capabilities identified earlier. Using techniques from the Case Based Reasoning literature (Kolodner, 1993) we believe that it will prove possible to detect recurring causal patterns (e.g. dependency structures including formalized requirements and part-specifications linked through issues and options) and retrieve design decisions that have worked out well in the past. Likewise it should prove possible to pick out patterns that have led to design conflicts and proactively offer advise on their resolution. This is an area of research that can only be explored once reasonably complete records of some initial design projects have been captured. However, we believe it holds out significant promise as a novel and powerful kind of design support—one whose realization will address a major class of design support needs and add strong arguments further supporting ADCT's basic approach to tracking design history and rationale.

ACKNOWLEDGMENTS

This work has been funded in part by NASA under contract NAS1-99089, and by NIST under contract 50-DKNB-1-SB084. We have also benefited greatly from input from engineers in Raytheon Inc, both from conversations about their general work environment and requirements for ADCT, as well as from the opportunity to observe the "Microglider" design exercise. All opinions stated in this report are those of Stottler Henke Associates, Inc. and its employees.

REFERENCES

Ballinger, G.H. Banares-Alcantara, R. and King, J.M.P, (1993). *Using an IBIS to Record Design Rational.* ECOSSE Technical Report 1993-17, Department of Chemical Engineering, University of Edinburgh, Edinburgh, Scotland.

Conklin, J. and Begeman, M.I. (1988). gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6, 4: 303-331.

Forbus, K. D., and de Kleer, J. 1993. *Building problem solvers*. Cambridge, MA: MIT Press.

Grudin, J. 1994. Groupware and Social Dynamics: Eight Challenges for Developers. *Communications of the ACM*, 37, 1, 92-105.

Kolodner, J.L. 1993. *Case-Based Reasoning*. San Francisco, California: Morgan Kaufmann..

Kunz, W., Rittel, H. *Issues as Elements of Information Systems*. Working Paper 131, Institute of Urban and Regional Development, University of California at Berkeley, 1970.

Petrie, C. (1993). The Redux' Server. Proceedings of the International Conference on Intelligent and Cooperative Information Systems (ICICIS), Rotterdam, May.