Semi-Formal Representations in Support of Design Coordination

Eric A. Domeshek

Stottler Henke Associates, Inc.
280 Broadway / 1st Fl
Arlington, MA 02474
domeshek@shai.com

Abstract

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*. By combining now-standard web-based enterprise information technologies with ideas and techniques from artificial intelligence we have produced a tool that supports capture of design history and rationale, and then exploits the resulting dependency network to identify possible consequences of design changes and conflicts. The system can generate targeted notifications of affected designers, and supports on-line discussions aimed at resolving design difficulties. In addition to helping ease team coordination problems, this tool can also provide a solid base for proactive design knowledge management. A version of this system has been implemented, and the resulting Advanced Design Coordination Tool (ADCT) is available for evaluation and as a customizable product.

Introduction: Problem and Solution Sketch

Over the last decades, information technology (IT) has 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 tool can also provide a solid base for proactive design knowledge management.

Five general trends in modern engineering practice motivate and shape the work reported here: (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 make team coordination a more important and challenging problem. They also place demands on any enterprise information management system supporting engineering design.

In addition to the general trends just listed, there are several other common, though not universal, aspects of design environments that place significant extra 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. Engineering projects with these properties place an additional premium on comprehensive and long-term knowledge management.

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 sketch of both the demands and constraints on such design support software, we note the following 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.

In light of these facts of life, our choices of architecture and supporting technologies should not be surprising. We adopted a three-tier architecture using commodity/COTS components and open (web-based) standards. For the client tier, we target COTS web browsers (e.g. MS Internet Explorer 5.x and higher, and Netscape Navigator 6.x and higher) driven with HTML and lightweight JavaScript. For our application tier we chose an open-source Java web-server (Jetty) with custom servlet extensions. For our back end, we chose an SQL relational database (Oracle 8i) to manage our data, including persistence and security.

The final, and most unconventional, aspect of the technology underlying our system is the exploitation of techniques from artificial intelligence 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 data organization is based on a straightforward representation covering design processes, products, and decision-making; in addition, we support definition of domain-specific ontologies for detailed formalization of design assertions. It is the introduction of these AI-based techniques that allow us to provide active design coordination capabilities.

System Metaphor

We set out 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, 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: a physical bound paper notebook, a loose-leaf binder, a rack of binders kept in a central location, an on-line directory structure of files, or, in a few cases, an on-line database.

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 from front-line engineers about the introduction of PDM systems in their organization.

Based on this input, and on 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*. We offer an easy entry route into the system: log in through a web browser, click to create or open a notebook, then click and type to add new notes to a book. Each note is 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.

System 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 promote capture not just of design history, but also of design rationale. The system provides notes to structure arguments and decisions behind assertions. When you 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. 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 formulae referring 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 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 can ease detection of design conflicts, and allow exploitation of the dependency

network to find those engineers implicated in conflicts, 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.

The next section provides detail on the implementation of our ideas, focusing on design representation, and notification generation, but including discussion of other system features such as discussion groups and versioning.

The Advanced Design Coordination Tool

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 design support IT issues sketched in the Introduction. During the system's development, we worked especially closely with engineers from Raytheon's Knowledge Center in their Missile Systems Division. The examples in this section reflect the current state of ADCT (v3.2.3), and our access to data on the Raytheon "microglider" design exercise: an unmanned, unpowered surveillance platform.

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 has fourteen common note categories for products/processes; future versions will allow extensions. Figure 1 shows relationships among several types of notes:

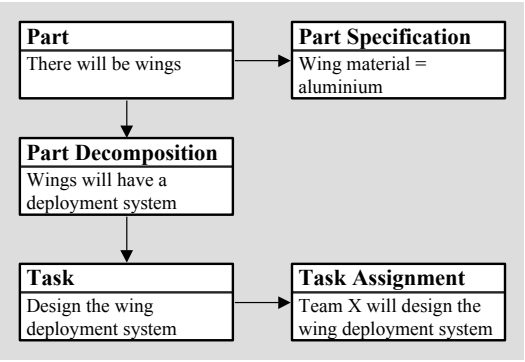

Figure 1. Sample Product/Process Notes

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.

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 3). 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 why a particular Option is chosen for some 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. This unmanned, guided glider had 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 2 shows a simplified version of the decision and rationale to have wings pivot forward. Figure 3 shows this Issue's Options, Criteria, and Evaluations in a grid layout.

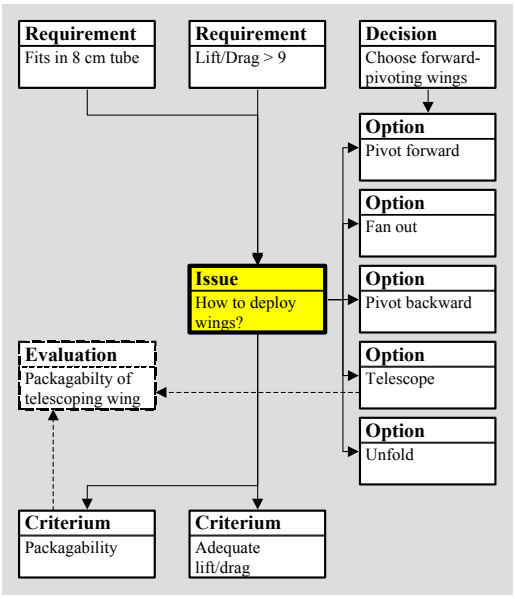

Figure 2. Rationale Notes for Wing Deployment Issue

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 3. Rationale Grid for Wing Deployment Issue

Linking "What" & "Why"

Product/process notes capture "what" is possibly be true about the design or design process, while rationale notes capture "why" decisions were reached. To complete this picture, these two kinds of notes need to be linked. 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.

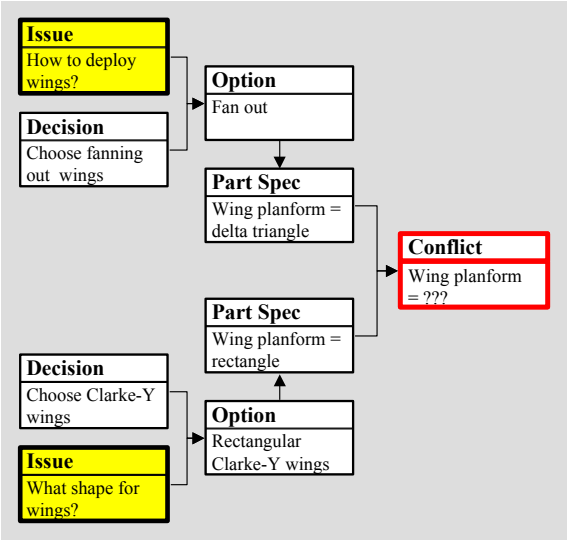

Figure 4. Incompatible Decisions Leads to Conflict

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 4 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.

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 automatically routes change notifications to appropriate team members. Such notifications trigger a flashing icon in the user's display. Users can review pending notifications, and easily jump to the relevant Notes. 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 in 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 support it, and the authors of those Options' Decisions are notified of the Conflict. The decision-makers are subscribed to a discussion group on 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 explorations of alternate designs ("what ifs") 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 public comments. ADCT stores the history of such discussions along with the basic Note.

Conflict discussions are special: they are automatically launched with 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 in the dependency network, 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

With the general demise of the old-style waterfall design method, 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. We are currently working on exploiting this versioning capability to support incremental extensions to the system's underlying design ontology on a per-project basis; the export capability then will facilitate ontology merging for evolution of a comprehensive installation-wide ontology.

Formalization & Proactive Case Retrieval

We have ambitious plans for ADCT's future development. Here we focus on a cluster of issues requiring further application of AI technology. Inclusion of capabilities for developing and applying ontologies make more sense as more use is made of the formal assertions. Example uses include detecting and avoiding duplication of content, detecting and reporting conflicting assertions, and automatically interoperating with formal analysis tools.

One additional way increased formalization can help—a way that represents a promising and powerful application of the materials accumulated in ADCT—is using formal design records as a mine for reusable design fragments and lessons learned. Using techniques from the Case Based Reasoning literature (e.g. Kolodner, 1993) we believe that it will prove possible to detect recurring causal patterns 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 advice on their resolution. Once we have reasonably complete records of some initial designs we will begin to explore this direction for exploiting ADCT's data.

References

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..

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