Capturing Design Rationale in Concurrent Engineering Teams

Mark Klein, Boeing Computer Services

The few existing systems that capture the rationale behind design decisions are severely limited. This new prototype offers an integrated and generic framework with much broader capabilities.

Output from the design of an artifact typically includes blueprints, CAD files, manufacturing plans, and other documents that describe the result of a long series of deliberations and trade-offs by the members of concurrent engineering (CE) teams. The underlying intent and logical support (that is, the *rationale*) for the decisions captured in these documents is usually lost or, at best, represented in a scattered collection of paper documents, project and personal notebook entries, and the recollections of the artifact's designers. This information can be very difficult to come by, and its representation is such that computers can provide little support for managing and utilizing it.

Intensified global competition and increasingly complex artifacts are making it more critical to capture the design rationale in a highly usable form. The potential benefits are manifold. An explicitly represented rationale can help individual designers clarify their thinking, and let all team members critique and augment the reasoning behind decisions.[1-5] Rationale capture helps identify design changes as well as the causes and potential resolutions of conflicts between designers.[6] It documents design decisions for new team members, new designers, and artifact users.[7] Existing designs that address similar requirements can be retrieved, understood, and modified to meet current needs. Perhaps more importantly, subsequent CE teams can use the rationale in their design activities.

To achieve these benefits, however, significant challenges must be met. The representation must allow designers to express their design reasoning in a natural way; at the same time, it must be formal enough to support useful computational services. Since CE teams include multiple participants working on overlapping aspects of the design, the representation must support concurrent editing. In addition, the process of describing rationale should impose the minimum possible overhead on the design process.

Most existing rationale-capture approaches support only individual users and are thus not suited to team contexts (though Yakemovic and Conklin[1] and Lee and Lai[2] describe some exceptions). More importantly, they capture the rationale for decision-making in general, but not for design decisions in particular. They simply add yet another document to the set produced by existing design tools, as shown in Figure 1.

0018-9162/93/0100-0039$03.00 © 1993 IEEE

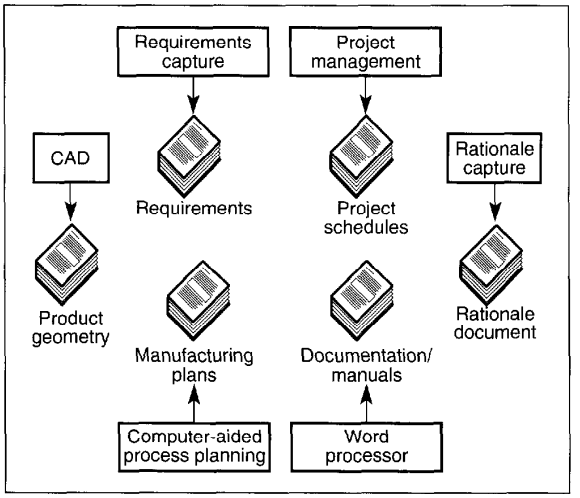

Figure 1. Rationale captured as a distinct document.

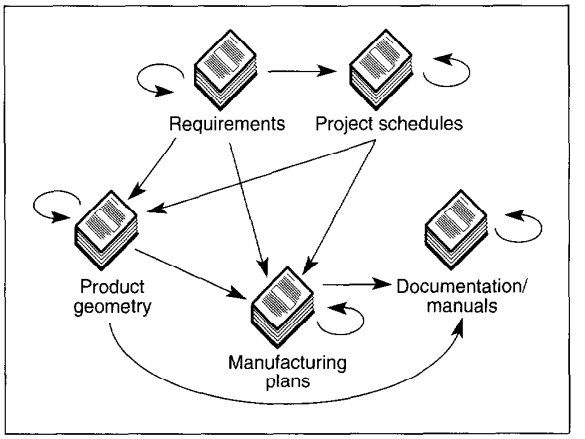

Figure 2. Rationale as decision interdependencies.

Design reasoning model

Underlying the DRCS rationale language is a model of how designers think. Rationale is essentially a record of the reasoning process an individual used to reach certain conclusions. Hence, a description language expressed in terms that accurately mirror the individual's reasoning process will be easier to use. A rationale language for the medical domain, for example, would be much less useful if it did not include terms like "hypothesis," "evidence," "symptom," and so on, since these are entities used in medical reasoning.

Central to a design reasoning model is, of course, how the design itself is represented and refined. This representation includes both the physical artifact produced and the plans (that is, temporal artifacts) followed to define and actually produce it. In DRCS, physical artifacts are viewed as collections of modules, which can represent entire systems, subsystems, or their components. As shown in Figure A, each module has its own characteristic attributes, whose interfaces (which have their own attributes) have a given type of connection. The resources that a module uses, such as cost and weight, are represented using a special class of attribute.

A computer, for example, can be described as a set of VLSI chip modules with attributes describing their functionality, power consumption, and so on. The connections between module interfaces (pins) are realized as (electrical) deposited wires. At another level, we can view an entire board as a module connected directly to the bus and indirectly to other boards. The interfaces and connections at this level describe the data and control protocols between these systems. Similarly, hydraulic systems can be viewed as collections of pipe, switch, tank, and pump modules linked via hydraulic connections (for example, threaded pipe).

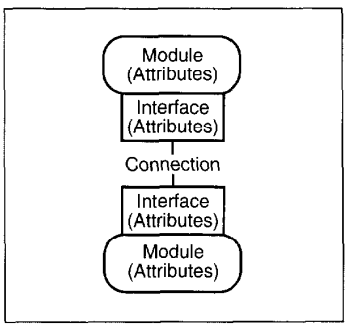

Figure A. Design description scheme.

In the DRCS model, artifact descriptions are refined using an iterative least-commitment synthesize-and-evaluate process. An artifact description starts as one or more abstract modules representing the desired artifact (for example, "airplane," "computer," or "software application") with specifications represented as desired values on module attributes (for example, "passenger capacity should be > 350"). This is refined into a more detailed description by constraining the value of module attributes, connecting module interfaces (to represent module interactions), decomposing modules into submodules, and specializing modules by refining their class (Figure B).

If we were designing an airplane, for example, we might decompose the top-level "airplane" module into wing, tail, and body section modules as well as electrical, hydraulic, and mechanical subsystem modules. Interactions between modules (for example, physically connected components) are represented as connections between module interfaces.

Plans are viewed as (perhaps partially) temporally ordered collections of tasks (Figure C). The tasks include associated attribute constraints. An artifact production plan would thus be represented as a sequence of tasks corresponding to operations such as machining, inspection, and the like. Every task includes one or more primitive actions that actually implement the task.

Plans, like artifacts, are defined in an iterative least-commitment manner. The essential difference is that the basic

Such approaches have limited expressiveness and therefore limited computational usefulness. The corresponding rationale-capture tools provide spotty capture of design rationale and may generate descriptions that are inconsistent with the design descriptions. Designers can waste their time on issues that later prove unimportant, because current rationale-capture tools do not let them focus on issues revealed by actual inspection of the evolving design description.[8]

Overcoming these limitations requires systems that let CE team members conveniently describe the dependencies between the decisions captured by existing design tools. Figure 2 illustrates this idea. It shows, for example, that the rationale for a product geometry decision consists of the requirements it attempts to satisfy, the time limits for design dictated by the schedule, and the other geometry decisions it logically depends on. Similarly, a manufacturing-plan decision is justified in terms of supporting decisions that involve, for example, project schedule and product geometry.

While systems that integrate design and rationale representations do exist (see Fischer et al.[8]), their design representations are highly domain specific (for example, kitchen design) and do not easily generalize to other domains. The fundamental challenge, then, is to provide an *integrated* and *generic* framework for capturing rationale in team contexts.

DRCS is a design rationale capture system that meets this challenge. Its underlying rationale language is based both on previous work in decision-rationale capture and on a generic model of design reasoning (see the sidebar "Design reasoning model"). The language is designed to capture all important aspects of design decisions and their interrelationships in a natural way. DRCS itself explores how to enhance design systems so that they will support collaborative editing of designs and their rationale.

entity is a task rather than a module, and tasks are temporally ordered rather than connected via interfaces. For both physical and temporal artifacts, DRCS provides a constraint language that allows indefinite descriptions and thus least-commitment design. A constraint language is a common approach to supporting conflict avoidance and early conflict detection.[1,2]

In parallel with the iterative refinement of the design description, the design is evaluated with respect to how well it achieves the specifications. Based on this analysis, we may choose to select one design option over another or to modify a given option so that it addresses an identified deficiency. The stages of specification identification, design option definition, evaluation, and selection/modification can be interleaved arbitrarily throughout the design process.

In addition to reasoning about the design itself (that is, reasoning at the domain level), designers also reason at the metalevel about the process they use to define the design.[1] A designer may have a plan for how to create the design. The plan might include tasks such as "collect requirements," "develop options," and "perform trade study." If several design options are available, a designer may reflect on which option to select. If a conflict between two or more design goals and actions occurs, the designer must resolve the conflict. The design-reasoning process is generally goal driven, in the sense that actions are taken as part of strategies intended to achieve goals such as meeting a specification, refining a design option, making a control choice, or resolving a design conflict.

This generic model of design reasoning is based on classical systems engineering as well as AI models of artifact planning[1] and design.[2,3] These models have been applied successfully to a wide variety of domains including electrical, electronic, hydraulic, and mechanical systems, as well as software.

References

1. M.J. Stefik, "Planning With Constraints (Molgen: Parts 1 and 2)," *Artificial Intelligence*, Vol. 16, No. 2, 1981, pp. 111-170.
2. M. Klein, "Supporting Conflict Resolution in Cooperative Design Systems," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 21, No. 6, Dec. 1991, pp. 1,379-1,390.
3. C. Tong, "AI in Engineering Design," *Artificial Intelligence in Engineering*, Vol. 2, No. 3, 1987, pp. 130-166.

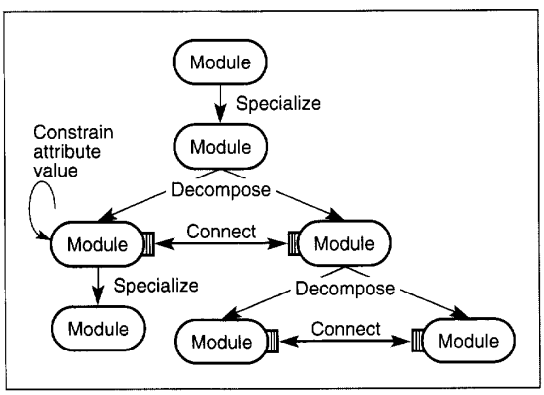

Figure B. Design refinement process.

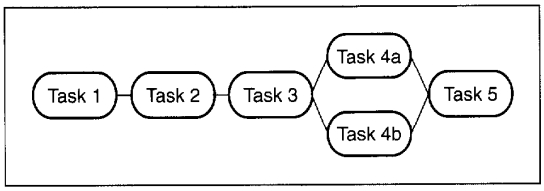

Figure C. A plan description.

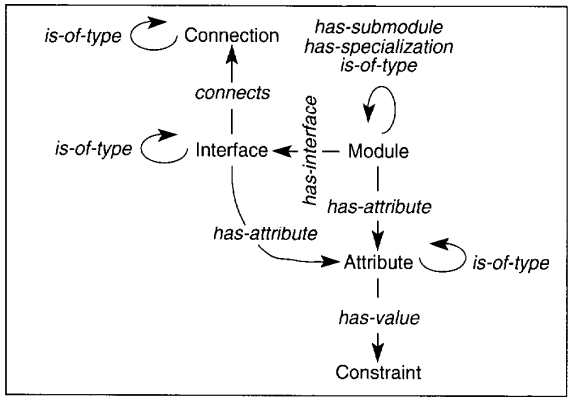

Figure 3. Artifact synthesis entities and relationships.

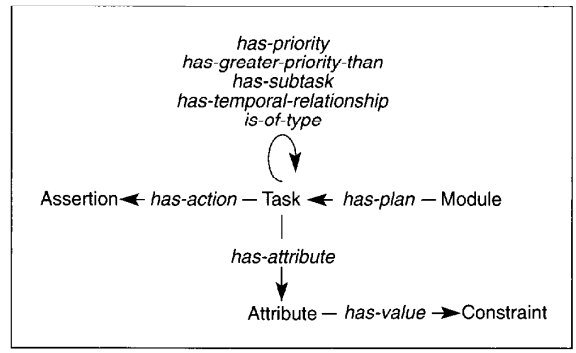

Figure 4. Plan synthesis entities and relationships.

DRCS rationale language

The DRCS rationale language uses a vocabulary of assertions to capture design reasoning. The assertions consist of *entities* such as modules, tasks, specifications, and versions, as well as *claims* about these entities.

Claims come in two main types. A predefined vocabulary of *relation* claims describes relationships between assertions. Any claim can serve as part of the rationale for another claim. Hence, we can make claims about the design (for example, module-1 *has-submodule* module-2), claims describing the rationale for design decisions (for example, value-1 *is-derived-from* procedure-1), claims concerning why we should believe this rationale (or not), and so on recursively. There is also an all-purpose text claim for capturing information not otherwise expressible.

The net result of describing designs and rationale in this way is a graph of entity- and text-claim instances connected by relational claims. The following discussion of the vocabulary of claims and entities that make up the DRCS rationale language divides the language into five components.

Synthesis. This language component captures the actions used to define artifacts and their plans. Figure 3 shows the language entities relevant to artifacts. (In this and the following figures, primitive language entities appear in plain font, while relation claims appear as directed arcs with italic labels. The latter signify that the given relationship can hold between the entities at the arc source and target. Entities with the "is-of-type" relation have an associated type taxonomy that a user can select from.)

The basic entities for artifact description include modules, attributes, interfaces, and connections. Modules can have submodules or specializations. Attributes can have values, expressed using a constraint language. (Constraint languages express indefinite descriptions and have been used extensively in numerous design and planning systems.[9])

Examples of DRCS in use

Imagine some designers working on the preliminary design for a new airplane. Figure D shows some of the initial specifications and commitments: The final airplane will cost less than $10 million, have a turning radius of less than 180 feet, and so on. It will be made of either aluminum or graphite, have a passenger capacity that is a given function of its length, and consist of interconnected

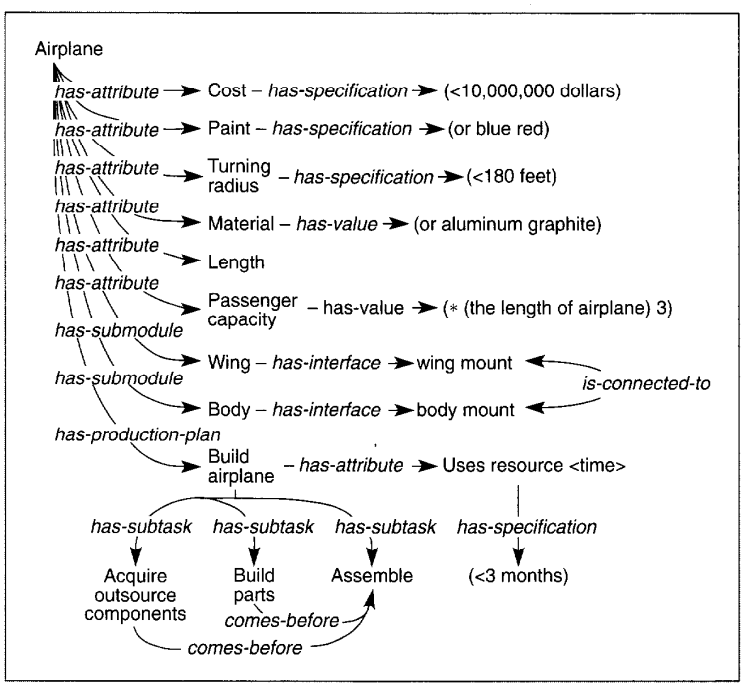

Figure D. Example of initial design specs and commitments.

The DRCS constraint language provides a wide range of constructs including absolute constraints such as inequalities, ranges, and sets as well as relational constraints such as Boolean and mathematical equations.

Figure 4 illustrates the capture of plan descriptions. The basic entities here are tasks. Plans to produce an artifact are related to the artifact's top-level module via a "has-plan" claim. Every plan is represented as the hierarchical decomposition of a top-level task into temporally ordered subtasks with associated primitive actions. The DRCS language captures this decomposition using "has-subtask," "has-action," and "has-temporal-relationship" (for example, "comes-before" or "comes-after") claims. Task actions can be any assertion. Plans can have priorities.

For both artifacts and plans, an important kind of attribute is the "uses-resource" attribute. Defining this requires specifying the type of resource as well as the amount used. Resources can include time, weight, money, tools, people, and so on.

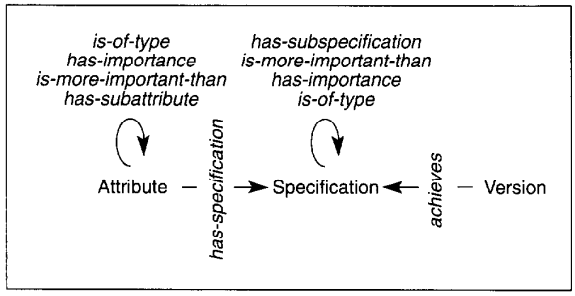

Figure 5. Evaluation entities and relationships.

Evaluation. The evaluation component of the DRCS rationale language captures not only design specifications but also how well they have been achieved. Figure 5 illustrates the evaluation entities and relationships. Design and plan specifications are defined as desired values for design and plan attributes. Attributes and specifications can have different types, priorities, and subsumption relationships. Attributes and specifications can have types. Specification types include objectives, requirements, and preferences. How critical these elements are differs from one specification type to another, and thus by implication, so

wing and body submodules. The manufacturing process should take no more than 3 months per plane and consist of partially ordered subtasks including "acquire outsource components," etc.

As the design process continues, the designers begin to address achieving the turning-radius specification. One possible strategy is to use folding wings. Figure E illustrates how the language captures this reasoning: The designers raised the decision problem of figuring out how to achieve the turning-radius specification, proposed a strategy to do so, and took actions (in this case, specializing the wing module) with the intent of implementing this strategy.

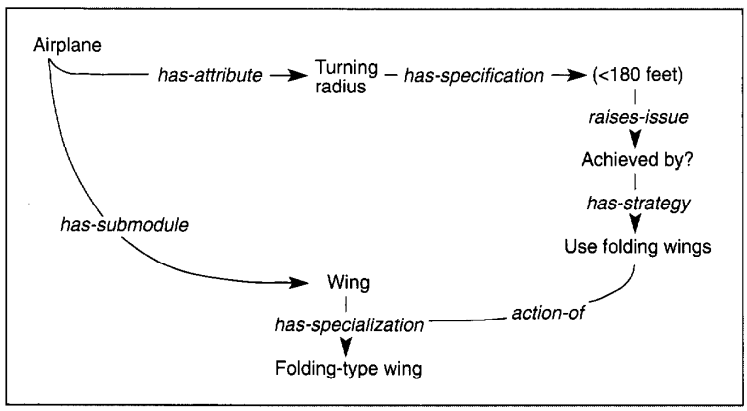

Figure E. Rationale for folding wing tips.

The specification concerning the airplane turning radius, however, turns out to be controversial. Figure F captures the line of pursuant argumentation: A designer asks what the turning radius of existing big planes is, then claims that the new airplane need do no better. Should the new airplane's original specification be replaced by a less stringent one, the designers can use the rationale graph to determine what derived decisions, such as the choice of folding wings, potentially need to be reconsidered.

DRCS can also represent the rationale for metalevel decision making, for example, deciding to try a particular option at a choice point or to resolve a conflict in a particular way. In both these cases, an assertion representing the decision problem (for example, resolved-by for conflicts or is-the-best-option-for for choice points) is linked via a strategy to the actions (for example, creating a new design option for conflicts, or choosing an existing one for choice points) that address the decision problem.

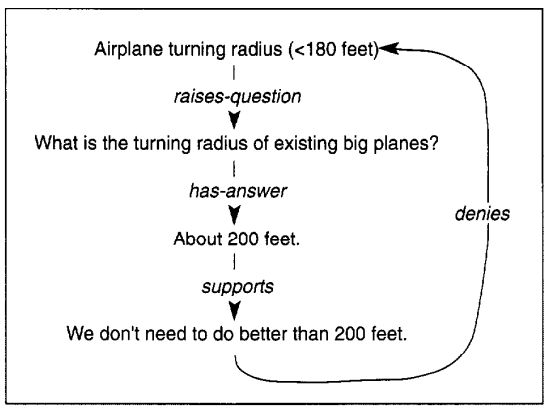

Figure F. Argumentation concerning the turning-radius specification.

does our willingness to relax them. We can say that a design version achieves a given specification; precisely how is described in the rationale for that claim (see the discussion under "Versions").

Intent. When taking some kind of design action, a designer is usually pursuing a strategy to find an answer for some problem; that is, the designer has some intent when taking that action.

Figure 6 illustrates the intent model for the DRCS language. Any assertion in a design description can raise a decision problem. There is, in fact, a preenumerated set of decision problem types — one for every relation defined for a given assertion type. For example, there is a decision problem for the "has-submodule" relation on modules (where the problem is determining how to decompose the module), a decision problem for the "has-value" relation on attributes (where the problem is determining what the attribute value should be), and so on. Some decision problems can have greater priority than others. The strategy used to address a decision problem is represented as a "has-strategy" link to the top-level task of a plan.

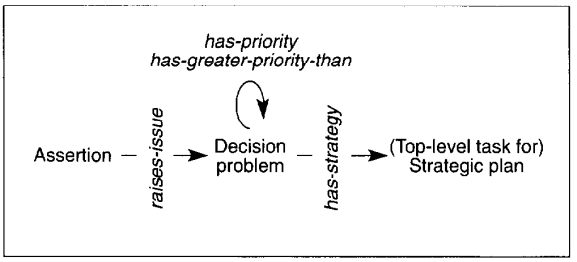

Figure 6. Intent model entities and relationships.

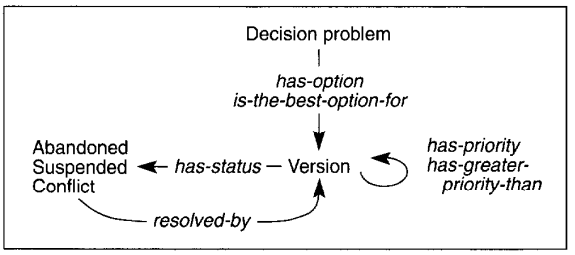

Figure 7. Versions model entities and relationships.

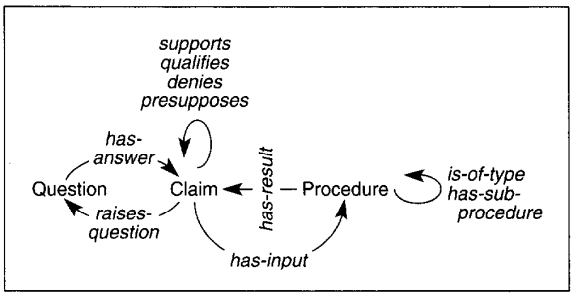

Figure 8. Argumentation model entities and relationships.

Versions. The versions model, illustrated in Figure 7, captures how the designer creates and explores the space of design alternatives. The designer creates new versions whenever tentative decisions are defined and/or alternatives are explored, that is, whenever options are defined for a decision problem. Every option for a given decision problem is asserted in a different version. The versions storing the options can have differing priorities as well as statuses. If a given version has the status "conflict," we can indicate which alternate version resolves that conflict. The preferred option for a decision problem is represented by an "is-the-best-option-for" claim.

Argumentation. The fifth component is the argumentation model, illustrated in Figure 8. It describes the reasons for and against believing claims. The basic entities include both relation and text claims as well as procedures and questions.

Claims can support, qualify, deny, or presuppose one another. Designers can use the "has-result" and "has-input" claims to link claims to the procedures used to derive them and the inputs to those procedures. Procedures can be mathematical equations or less structured information such as textual reference sources, handbooks, catalogs, and standard engineering tables. An individual can raise "questions" about the validity of a claim and assert that given claims answer these questions. Any synthesis, evaluation, intent, versions, or argumentation claim can itself be the subject of argumentation claims.

Advantages of DRCS language. The DRCS design rationale language is an extension, with substantial modification, of previous work in decision-rationale capture. Its main contribution is to integrate the rationale language with a generic design-description language applicable to a wide range of design domains. This approach offers a number of advantages over previous work.

First, the DRCS language is more expressive. Generic decision-rationale languages such as gIBIS[1] and DRL[2] use natural-language text to describe the requirements, decision problems, and options. By contrast, DRCS uses a structured language with explicit semantics. It describes requirements as a desired attribute value and design options as interconnected modules and tasks; it selects decision problems from a preenumerated set with known semantics.

Second, DRCS can capture design rationale on the basis of programmatic concerns. For example, it can use links between plan resource limits and design attributes to keep the design definition or manufacturing process from being too resource intensive. DRCS also incorporates a model of intent — something absent from most decision rationale work. For example, while DRL includes a "goal" entity, it provides no way to link goals to the strategy for achieving them and to the actions that implement the strategy. The DRCS information-theoretic content is thus significantly higher, but not at the cost of being domain specific (as it is, for example, in Janus[8]).

Third, DRCS's explicit semantics increase the possibilities for computational support. Since the problem and specification semantics for decisions are known, for example, it is much easier to fetch previous design cases that dealt with similar challenges. We can more easily find the differences and similaritics between candidate design versions by identifying how the designs diverged and

why. Integrated design/rationale capture supports conflict detection, classification, and resolution.[6] Controversial design decisions can be searched by looking for underlying claims that include many instances of support and denial. Users can determine the consequences of withdrawing a design choice by deleting all derived decisions without independent support; they can review the options explored for a given problem by checking all the has-option claims stemming from the decision problem, and so on.

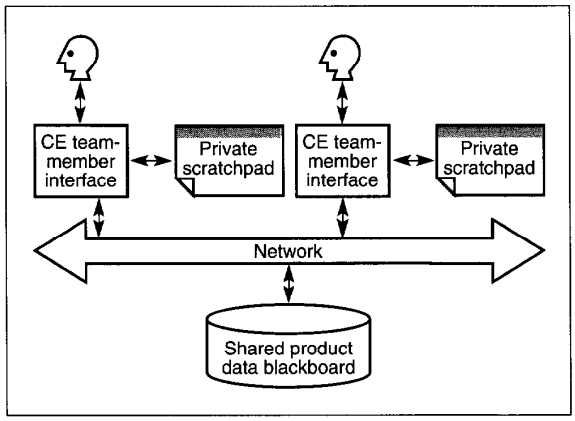

Figure 9. DRCS architecture.

Finally, DRCS is more natural than most previous languages for describing design rationale. Designers can attach the rationale directly to the design aspect it refers to (module decomposition, attribute value, etc.), rather than to a piece of text. In DRL, specifications are represented as subgoals of decision problems, and a new copy of this subgoal must be created for every decision problem affected by the specification. DRCS represents specifications simply as desired values for module attributes.

DRCS

Current design tools, as noted earlier, do not in general support rationale capture as an integrated component of their operation, nor do they support groups as opposed to individuals. DRCS was developed to improve understanding of how existing design systems can be augmented to address both these purposes.

DRCS is currently implemented in Common Lisp on several networked Symbolics workstations. Figure 9 illustrates its architecture. CE team members receive interfaces that let them view the design and rationale information on a shared blackboard. They can also make changes on their private scratchpads and "publish" the scratchpad contents so that the contents update the blackboard. Users can publish their changes as they are made, allowing essentially real-time collaborative editing, or they can choose to publish them periodically.

DRCS provides CE team members with a direct-manipulation graphical interface, shown in Figure 10. Users can create windows that present a subset of the design data from many possible per-

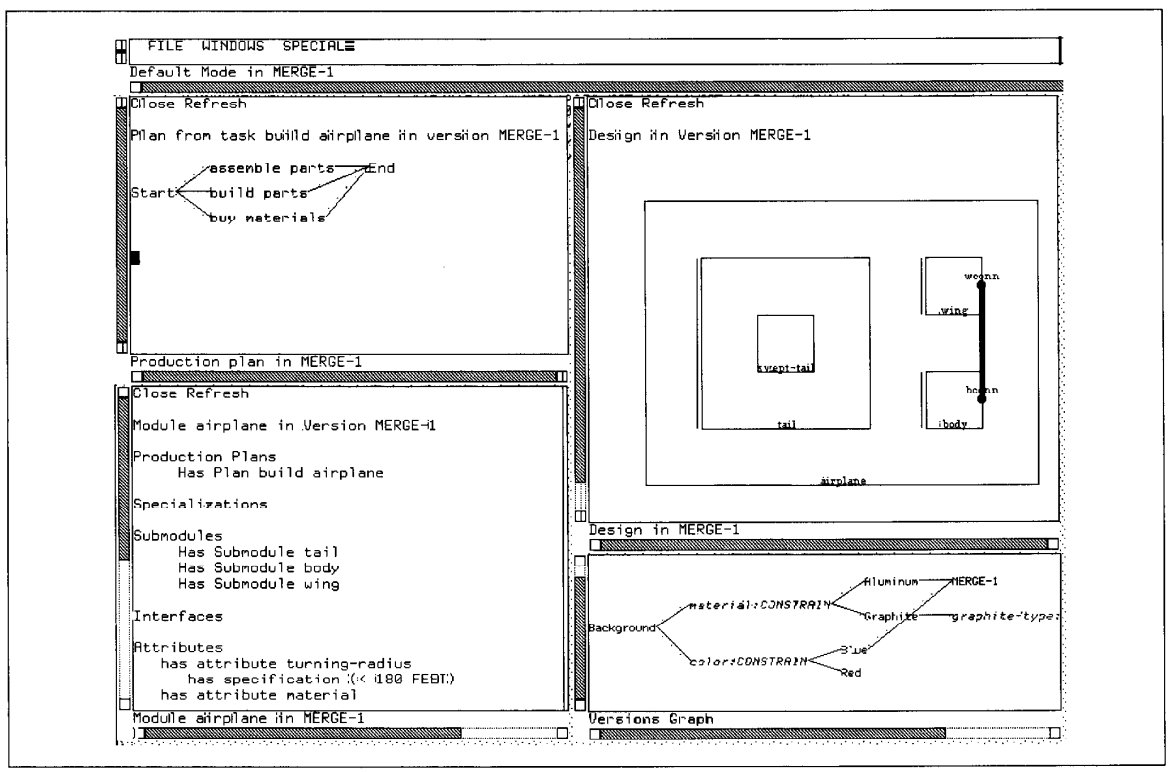

Figure 10. Example of the DRCS interface in use.

spectives, each highlighting different aspects of the design/rationale description. For example, one window can display the current artifact design as rectangular modules with lines representing connections. Another can show PERT charts of the ordering of tasks in a given plan. Another can graph the argument structure affecting a given claim or present the current set of versions as a lattice, and so on. These windows dynamically update themselves whenever the product-data subset they view changes, so they are continuously up to date. In addition to simply displaying product data in some predefined format, users can create instances of "analysis" windows that present information such as pointers to circular or incomplete argument structures or to questionable decision choices.

The topmost window in the Figure 10 display contains the menubar. Each item in it produces a menu of options when selected. The File menu options include saving the current state of the user interface and publishing the user's private scratchpad. The Windows menu supports the creation of new perspective windows or the ability to cycle through existing ones, while the Special menu lets users create analysis windows. Other views in Figure 10 are the versions graph (lower right), the artifact design in one of the versions (upper right), a description of one component in that design (lower left), and a description of the plan used to produce that component (upper left). The printed representation of every entity and claim is mouse sensitive: When the pointing device clicks on it, the system displays a menu of options that make sense in the context of that assertion.

There are two classes of options for any assertion: perspective creation and editing. Perspective creation options generate windows that view the assertion from a given perspective. When clicking on a plan's top-level task, for example, users can create windows that view it either as temporally ordered leaf tasks or as a task decomposition hierarchy. Editing options let users update the design rationale database; for any assertion, the menu will include a list of all the types of claims possible to make about that assertion. To add an attribute to a module, for example, the user clicks on the module and selects the "Define Attribute" option. A prompt then asks for an attribute name, and DRCS creates an instance of that attribute connected to the module via a "has-attribute" claim.

DRCS integrates design-decision and design-rationale capture in a single tool.

To connect two module interfaces, the user selects the "Make Connection" option for one interface and then selects the interface to connect to. To add support for a claim, the "is-supported-by" option is selected for that claim and then either an existing claim is selected or a new one is created. DRCS automatically creates the appropriate "supports" relation between these claims. A few mouse operations usually describe design decisions and their interdependencies (that is, their rationale), though text entry is sometimes required.

The DRCS interface builds upon rationale-capture systems such as gIBIS,[1] Sibyl,[2] and Janus,[8] and to a lesser extent on hypertext systems without an explicit rationale language (for example, see Uejio[10] and Lakin et al.[11]). The key difference between DRCS and these systems is that DRCS integrates in a single tool a general approach to both design-decision and design-rationale capture. Users thus have no need to switch tools when describing the design as opposed to its rationale. They can attach rationale directly to the design claims of interest and focus their efforts on describing rationale that the evolving design description reveals as critical.

Future directions

DRCS was developed to explore how current rationale-capture approaches can be extended to provide more effective support for the capture of computer-interpretable rationale from multifunction CE design teams. It has been used successfully to record rationale for a variety of simple new designs and to re-represent information from more complex existing designs. The resulting information has supported computational services that generic decision-rationale representations could not support. This experience shows that the DRCS approach is valid.

There are, moreover, rich possibilities for future growth. The rationale-capture language must be augmented to capture geometric information (by incorporating a feature-based geometric representation) and tentative or "fuzzy" argumentation.[2] In addition, better methods must be developed to allow effective display and use of anticipated large and highly complex product data/rationale networks.

Rationale-capture systems impose significant overhead on the design process. This is exacerbated by the fact that the people who benefit from rationale capture often are not those who are asked to perform it. The challenge is to make the cost/benefit ratio attractive to the individuals asked to enter rationale.

While the point-and-click interface metaphor reduces DRCS's overhead, we need to do more. In addition to maximizing its current services to users, DRCS may add support for rationale capture via English text. This is less daunting than one might imagine: Probably only a limited subset of English is needed to express design rationale, and the current design context can help reduce the semantic ambiguity of natural-language text.

DRCS is currently a stand-alone research prototype. To have significant impact, the technology must be added to the decision-capture tools actually used by CE design teams. This requires advances on several fronts. Current product data and/or interapplication link standards must be augmented to include a design rationale representation. This will involve both adding rationale-description primitives to current standards and defining the mapping between the existing and DRCS product-data representations. CE team-member interfaces must be updated to let users collaboratively view, edit, and link different kinds of shared product data.

One approach is to enhance existing design tools so that they can provide additional product-data display formats (for example, the addition of manufacturing-data displays to a CAD tool) and can also allow linkages among this data. Another approach is to provide support at the operating-system level, in effect extending the cut-and-paste metaphor

used in the Macintosh operating system to support creation of cross-application rationale links. Both approaches are currently under evaluation for viability in Boeing's computing context. ■

References

1. K.C.B. Yakemovic and E.J. Conklin, "Report on a Development Project Use of an Issue-Based Information System," *Proc. CSCW 90*, ACM Press, New York, 1990, pp. 105-118.
2. J. Lee and K.Y. Lai, "What's in Design Rationale?" *Human-Computer Interaction*, Vol. 6, Nos. 3-4, 1991, pp. 251-280.
3. R. McCall, "PHIBIS: Procedurally Heirarchical Issue-Based Information Systems," *Proc. Conf. Planning and Design in Architecture*, ASME, Boston, 1987, pp. 17-22.
4. A. MacLean et al., "Questions, Options and Criteria: Elements of a Design Rationale for User Interfaces," *J. Human-Computer Interaction*, Special Issue on Design Rationale, Vol. 6, Nos. 3-4, 1991, pp. 201-250.
5. J. Mostow and M. Barley, "Automated Reuse of Design Plans," *Proc. Int'l Conf. Eng. Design*, IEEE, Piscataway, N.J., 1987, pp. 632-647.
6. M. Klein, "Supporting Conflict Resolution in Cooperative Design Systems," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 21, No. 6, Dec. 1991, pp. 1,379-1,390.
7. R. Balzer, "Capturing the Design Process in the Machine," *Proc. Rutgers Workshop on Knowledge-Based Design Aids*, New Brunswick, N.J., 1984.
8. G. Fischer et al., "Making Argumentation Serve Design," *J. Human Computer Interaction*, Vol. 6, Nos. 3-4, 1991, pp. 393-419.
9. M.J. Stefik, "Planning with Constraints (Molgen: Parts 1 and 2)," *Artificial Intelligence*, Vol. 16, No. 2, 1981, pp. 111-170.
10. W.H. Uejio, "Electronic Design Notebook for the DARPA Initiative in Concurrent Engineering," *Proc. Second Ann. Concurrent Engineering Conf.*, CERC, Morgantown, W.Va., pp. 349-362.
11. F. Lakin et al., "The Electronic Design Notebook: Performing Medium and Processing Medium," *Visual Computer*, Vol. 5, No. 4, 1989, pp. 214-226.

Mark Klein is an artificial intelligence specialist in the Boeing Computer Service's Collaborative Computing Program, where his work focuses on collaborative problem-solving with human and machine-based agents. His research interests address distributed AI systems that use multiple kinds of potentially conflicting expertise to solve problems.

Klein received his PhD in artificial intelligence from the University of Illinois in 1989.

Readers can contact Klein at Boeing Computer Services, PO Box 24364, 7L-64, Seattle, WA 98124-0346; e-mail mklein@atc.boeing.com.

HIGH PERFORMANCE JOURNALS FROM ACADEMIC PRESS

Journal of Visual Communication and Image Representation

Editors-in-Chief
Yehoshua Y. Zeevi
Technion-Israel Institute of Technology, Haifa, and CAIP Center, Rutgers University, Piscataway, New Jersey

T. Russell Hsing
Bell Communications Research, Morristown, New Jersey, and CAIP Center, Rutgers University, Piscataway, New Jersey

Promoting Global Communication

The **Journal of Visual Communication and Image Representation** publishes papers on the state-of-the-art of visual communication and image representation, with emphasis on novel technologies and theoretical work in this multidisciplinary area of pure and applied research. The field of visual communication and image representation is considered in its broadest sense and covers both digital and analog aspects as well as processing and communication in biological visual systems.

Volume 4 (1993), 4 issues — In the U.S.A. and Canada: $142.00
ISSN 1047-3203 — All other countries: $171.00

Journal of Parallel and Distributed Computing

Allan Gottlieb, Editor, Hardware and Software Systems
New York University, New York City

Kai Hwang, Editor, Special Issues
University of Southern California, Los Angeles

Sartaj Sahni, Editor, Algorithms and Applications
University of Florida, Gainesville

Side by Side

Parallel and distributed computing systems, side by side, are improving the ability of computers to solve increasing numbers of difficult and complex problems as quickly and as efficiently as possible. The **Journal of Parallel and Distributed Computing** publishes original research papers and timely review articles on the theory, design, evaluation, and use of parallel and/or distributed computing systems.

Volumes 17–19 (1993), 12 issues — In the U.S.A. and Canada: $324.00
ISSN 0743-7315 — All other countries: $407.00

Increase your performance.
Subscribe to computer science journals from Academic Press.

Sample copies and privileged personal rates are available upon request.
For more information, please write or call:

ACADEMIC PRESS, INC., Journal Promotion Department
1250 Sixth Avenue, San Diego, CA 92101, U.S.A.
(619) 699-6742

All prices are in U.S. dollars and are subject to change without notice. Canadian customers: Please add 7% Goods and Services Tax to your order. S3239

Reader Service Number 2