Acquiring Design Rationale Automatically

Karen L. Myers   Nina B. Zumel
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
myers@ai.sri.com   zumel@ai.sri.com
650-859-4833   650-859-2539
FAX: 650-859-3735

Pablo Garcia
Innovative Product Engineering and Technologies
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
pgarcia@unix.sri.com
650-859-5419

Page Count: 45
Figure Count: 10
Table Count: 8

Acquiring Design Rationale Automatically

Abstract

The value of comprehensive rationale for documenting a design has long been recognized. However, designers rarely produce detailed rationale in practice because of the substantial time investment required. Efforts to support the acquisition of rationale information have focused on languages and tools for structuring the acquisition process, but still require substantial involvement on the part of the designer. This paper describes an experimental system, the Rationale Construction Framework (RCF), that acquires rationale information for the detailed design process without disrupting a designer's normal activities. The underlying approach involves monitoring designer interactions with a commercial CAD tool to produce a rich process history. This history is subsequently structured and interpreted relative to a background theory of *design metaphors* that enable explanation of certain aspects of the design process. The framework provides an environment that can acquire rich, meaningful rationale information in a time- and cost-effective manner, with minimal disruption to the designer.

Keywords: Design Rationale, Automated Acquisition, Plan Recognition, Clustering, Knowledge-based Methods

1 Introduction

Representations of designs in current-generation computer-assisted design (CAD) frameworks consist primarily of diagrammatic specifications, possibly augmented with simple annotations and *ad hoc* documentation. Even the most sophisticated design systems lack much in the way of structured documentation for the design: Why is the design the way it is? What key decisions and tradeoffs were made? What interactions and dependencies exist among components? What assumptions are critical for the success of the designed artifact?

It is well accepted within the design community that the availability of explicit, declaratively represented *design rationale* would be a tremendous asset. Design rationale would serve as a record of the basic structure of a design, codifying how the design satisfies specified requirements, as well as key decisions that were made during the design process. This information would facilitate collaboration among multiple distributed designers – a tremendous benefit for large-scale design efforts. Rationale would also provide guidance in exploring alternative designs, whether as part of the natural evolution of a design or in response to changing requirements. Finally, design rationale would enable both easier maintenance of artifacts over their life cycles and more effective reuse of designs by making it easier for downstream engineers to understand how a design works. For example, (Brazier et al., 1997) presents an example of stored rationale being used in the redesign of a model passenger aircraft to accommodate changes in the overall design requirements.

Despite the tremendous advantages that explicit design rationale would provide, designers rarely produce it in practice because of the substantial time commitment required. Tools that support the specification of structured rationale by a designer have met with limited success because they either demand substantial designer time to enter information (Carroll and Moran, 1991) or they change the manner in which designers work (Conklin and Yakemovic, 1991). Furthermore, designers have little motivation to participate in such activities since the benefits surface downstream of their contributions. Recently, nonintrusive approaches have been explored that involve video or audio recording of design sessions (Chen et al., 1991; Shipman and McCall, 1997); however, a lack of structure in the produced representations hinders effective use of the information that they provide.

Given the tremendous value of structured design rationale but the unacceptable burden of constructing it manually, we were motivated to explore the use of artificial intelligence (AI) methods for automatically generating rationale without disrupting the normal design process. Our work focuses on the *detailed design* phase, in which tools (*e.g.*, CAD systems, analysis packages) are used to generate a schematic that meets the specifications laid out for an artifact. This contrasts with the *conceptual design* phase, in which the scope and capabilities are set for the artifact to be designed.

1.1 Rationale: Conceptual vs. Detailed Design

Moran and Carroll (Moran and Carroll, 1996) describe the life cycle of an artifact as having a *requirements*, or conceptual phase; a (detailed) *design* phase, which results in a detailed specification of the artifact; a *building* (or *construction*) phase; a *deployment* phase, in which the artifact is marketed, distributed, and used; a *maintenance* phase; and possibly a *redesign* phase, in which the original design is modified to produce a new artifact.

The conceptual phase focuses on identifying and resolving high-level issues of functionality and requirements, with design rationale recording the justifications for the decisions that have been made. Decisions are grounded primarily in assumptions or nontechnical criteria such as end-user preferences. During detailed design, functionality and requirements may be refined. However, issues such as component structures and interactions, validation, functionality, and design alternatives are of greater importance. The level of abstraction at which these issues are considered is much lower than during the conceptual phase. Choices and decisions are grounded primarily in physical constraints on components and the designer's insights into the composition of good designs. These insights include criteria such as simplicity, ease of modification, and intuitive feel for success. Thus, rationale for the detailed design phase focuses on *how* a given design works, and *why* the specific detailed design choices were made.

Work to date on the acquisition of design rationales has focused on the issues and choices made during the conceptual phase: deciding on the functional requirements for an object, and the high-level design approaches to be pursued in meeting those requirements (Carroll and Moran, 1991). There has been little work on capturing the decision-making and logic that underlies the process of constructing the design itself. It has been shown that designers are reluctant to document their actions during the detailed design process (Fischer et al., 1991). Because of this resistance, it is critical to investigate nonintrusive methods for rationale acquisition during detailed design.

1.2 The Rationale Construction Framework

The premise for our work was the observation that many of the operations that a designer can perform with modern CAD tools (the *design substrate* (Fischer and Lemke, 1988; Hutchins et al., 1986)) have meaningful semantic content. For example, CAD tools allow users to select objects with assigned semantic types from predefined libraries. This contrasts with most tools for designing software, where interactions are generally at the level of keystrokes. Nonintrusive monitoring of the actions taken by a designer with a CAD tool would thus provide a rich, semantically grounded process history for detailed design. Techniques from AI could be used to structure this information into representations that would support query access and reasoning about *designer intent*. Valuable reasoning methods would include *clustering* techniques to aggregate CAD operations into abstract summaries of designer activity, *plan recognition* to identify key episodes of activity, and *qualitative reasoning* about the emerging design.

This paper describes the Rationale Construction Framework (RCF), which embodies the above ideas in a system that automatically constructs rationale information for the detailed design process. RCF records events and data of relevance to the design process, and structures them in representations that facilitate generation of explanations for designer activities. RCF operates in an opportunistic manner, extracting rationale-related information to the extent possible from observed designer operations. For RCF, we interpret rationale broadly to encompass any information that will further the understanding of a design and its development. This philosophy fits the *generative paradigm* (Gruber and Russell, 1992) for rationale construction, which focuses on supporting general queries about a design and its evolution rather than answering fixed sets of questions.

RCF extracts two different types of rationale-related information. The first is a series of hierarchical abstractions of the design history: *what* the designer did, and *when*. In addition, RCF reasons about intent – *why* the designer performed particular actions. A set of *design metaphors*, which describe temporally extended sets of designer operations that constitute meaningful episodes of activity, drives the extraction of rationale related to designer intent (Section 4.2). Design metaphors provide the basis for inferring intent on the part of the designer by linking observed activities to explanations for them. A rich query interface allows users to access extracted rationale from several different perspectives, enabling them to overview the design history, and to elicit information pertinent to their specific goals (Section 9).

Automatic generation of complete rationale for all aspects of a design is clearly infeasible. Certainly, designers make many critical decisions that are not explicit in the designs nor in the design process. The work reported here seeks to automate documentation of important but low-level aspects of the design process in a time- and cost-effective manner, thus freeing designers to focus their documentation efforts on the more creative and unusual aspects of the design. Ideally, the methods presented here would be complemented by interactive rationale acquisition methods that would enable designers to extend or correct automatically generated information. The ultimate goal is to produce an annotated design history that tracks dependencies, assumptions, and tradeoffs, and so would greatly facilitate the understanding of a design by downstream designers.

We evaluated RCF informally by applying it in a case study that involved the design of a three-degree-of-freedom surgical robotic arm.[1] RCF recorded designer activity over several versions of the arm, starting from a rough initial design, through various stages of refinements and optimizations. From these recordings, RCF was able to summarize designer activity at varying levels of abstraction, identify phases where the designer concentrated on various parts or subassemblies or where design parameters were tuned, track the results of design tradeoffs, and explain key design changes. The results validate the idea that meaningful rationale can be generated nonintrusively through application of appropriate AI techniques.

1.3 Different Perspectives on Rationale

As summarized by Shipman and McCall (Shipman and McCall, 1997), design rationale can be viewed (and defined) from different perspectives:

Argumentation In this view, rationale captures the process of reasoning and discourse, either by a single designer or a design team. Rationale consists of the problems or issues that arise in the course of a design, along with pros and cons for each alternative. Rationale as argumentation need not be passive, but rather can be used as a framework for structuring the design process, in order to clarify and facilitate it. For example, Conklin and Yakemovic (Conklin and Yakemovic, 1991) discuss the use of various implementations of IBIS (Rittel and Webber, 1973), or "Issue-Based Information System", in commercial design settings. They report that the use of these tools can often uncover key issues and communication breakdowns during the design process that might otherwise have been missed, or not detected until a later stage in the process.

Documentation From this perspective, rationale is viewed as a "decision trail" for use by observers outside of the design team. Rationale records what decisions were made, when, why, and perhaps by whom. Information would be presented in a way that makes the decision process clear to observers who do not have full background knowledge about the design effort.

[1] One of the authors served as the designer for this evaluation.

Communication Rationale is interpreted as the passive recording of design discourse as it occurs. Rationale stores the information needed to answer questions about what was done, and why, possibly for recall during the design process itself or for subsequent decision-making.

A rationale acquisition system can adopt more than one of these perspectives. The work described in this paper focuses on rationale as documentation and communication. Specifically, in recording the design process we wish to make explicit the relationship between design decisions and design requirements, as well as the interactions among various assemblies and components in the design. This information will be useful during the development of the initial design, providing the means for a designer to keep track of options that have been explored and reasons for certain modifications to the design. However, we envision that the extracted rationale will be of even greater value later in the life cycle of a design. We envision many of the potential users of RCF to be designers who have taken over a previous design for maintenance or reuse. For them, it is critical to be able to quickly develop an understanding of a design, including the key assumptions and tradeoffs that it embodies, to ensure that modifications are consonant with the intent of the original designer.

1.4 Overview of Paper

Section 2 describes the architecture for RCF, presenting overviews of the main components of the system. Section 3 summarizes the robotic arm design case used throughout to illustrate the workings of RCF. Section 4 presents an overview of our approach to rationale extraction, which is subsequently expanded in Sections 5 through 8. Section 9 describes mechanisms for accessing the extracted rationale. Section 10 discusses evaluation of the approach taken and future directions, while Section 11 describes related work. Section 12 presents our conclusions.

2 RCF Architecture

As depicted in Figure 1, RCF contains three main components: an enhanced CAD tool, the Monitoring module, and the Rationale Generation module (RGM).

Within RCF, the designer interacts with the CAD tool as if it were a stand-alone application. The operations that he or she performs, however, are tracked by the Monitoring module, which forwards relevant information about observed designer events in real time to the RGM. From this stream of events, the RGM constructs a *design event log* that provides a comprehensive history of

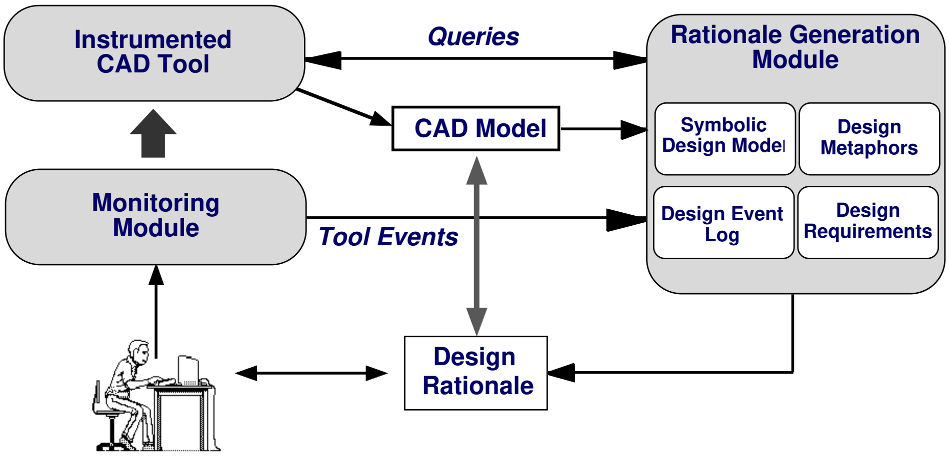

Figure 1: RCF Architecture

operations performed by the designer during the course of a design session. Based on the incoming events, RGM also constructs a *symbolic design model* that provides a qualitative description of the emerging design. The rationale construction methods exploit these two components to construct rationale, using *design metaphors* and background information about design requirements.

2.1 CAD Tool

Several criteria drove the selection of a CAD system upon which to build RCF. One criterion was the inclusion of sophisticated modeling capabilities that would provide a rich set of design operations suitable for demanding electromechanical design tasks. A second criterion was that the system employ a design substrate of sufficiently high level (Fischer and Lemke, 1988; Hutchins et al., 1986). Third, extensibility of the system would be critical for enabling both the insertion of monitoring hooks and the definition of additional CAD operations. After an extensive survey of available commercial and research systems, Bentley's MicroStation95 (Bentley Systems, Inc., 1995) was selected as the system that best satisfied our needs.[2]

To support rationale acquisition within RCF, the set of operations provided by MicroStation95 was extended to include several capabilities that raised the overall semantic content of its design substrate. One class of added operations, *annotations*, enables users to specify the *semantic type* of an object along with associated type-specific *semantic attributes*. The semantic type of an object refers to the intended semantic interpretation of the object with respect to the particular

[2]While the RCF rests on top of a particular tool, its underlying models apply more generally to a wide range of CAD tools.

design domain, in contrast to the structural and geometric description of the object. For example, the designer may declare that a given solid represents a gear, as well as specifying gear-specific attributes such as number of teeth, gear ratio, or quality. Such semantic information, which RCF uses extensively, is provided as a by-product of parametric design methods and part selection capabilities found in numerous state-of-the-art CAD systems. We also augmented MicroStation95 with a set of analysis programs that can be linked directly to components in the CAD model, thus extending the limited analysis capabilities within the core system. This modification reflects a growing trend toward building design environments that integrate a range of design tools. Finally, we added an ability to select components from predefined part libraries, which is standard in many current CAD frameworks.

2.2 Monitoring Module

The monitoring facility within RCF nonintrusively tracks designer operations within MicroStation95, generating a stream of *tool events* that are sent in real time to the RGM. Tool events are a system-dependent representation of the activity of a designer. The tool events extracted by the monitoring facility within MicroStation95 include the creation, deletion, and modification of points, two-dimensional profiles, parametric solids, free-form solids, boolean combinations of solids, and features on solid objects (including holes, bosses, protrusions, ribs, and cuts). Manipulation operations (*e.g.*, move, copy, rotate) are supported, as is the direct assignment of attributes. Creation and deletion of joints are possible, leading to the connection of parts in pairwise fashion into assemblies. The creation and importing of parts are supported, as is the invocation of built-in analysis programs. Finally, process-level commands such as the undoing and redoing of operations are also tracked. Overall, the set of monitored commands is adequate for a wide range of complex design tasks.

Certain aspects of the design process are explicitly ignored by the monitoring process. Examples include certain geometric information associated with the manipulation and definition of objects (*e.g.*, spatial positioning), and design commands not immediately relevant to rationale (such as viewing commands).

The monitoring process considers only fully prosecuted commands. Thus, intermediate changes made within the middle of specifying an operation are ignored. For example, an operation to connect two components with a specified type of joint requires selection of two objects and an appropriate joint type. For such multistep operations, a user may change the selected parameters for the operation before committing to a version of the command that should be executed. The RCF monitoring process tracks only the executed command.

2.3 Rationale Generation Module

The Rationale Generation Module (RGM), the main inferential component of the framework, performs the automated generation of rationale structures. The RGM incrementally constructs an abstract representation of observed CAD tool operations, called the *design event log*, which provides a tool-independent characterization of key design operations.

Based on the design event log, the RGM constructs a *symbolic model* of the emerging design in incremental fashion. This model contains the core elements of the design, along with key relationships among them (*e.g.*, part/subpart relationships, interpart constraints). Individual components are tagged with annotations that are relevant to rationale, such as timestamps, revision histories, and reuse information. The symbolic model of the design contains limited geometry information, restricted mostly to dimensional information for created objects.

The design event log and symbolic design model provide the evolving information base from which rationale information is generated, in conjunction with a formal specification of the requirements for the design task and a set of *design metaphors*. As described further in Section 4.2, design metaphors characterize sets of operations that constitute meaningful episodes of designer activity. As such, they provide the basis for inferring designer intent from designer actions.

2.3.1 Event Models

The operations supported within the design event model, while not exhaustive, were chosen for their adequacy with respect to an interesting set of design tasks. Two high-level categories of operations are distinguished within the system. *Base-level* operations (Table 1) support the direct creation, modification, and manipulation of objects. *Process-level* operations (Table 2) do not operate directly on design objects. Instead, they either manipulate information and metalevel structures related to the design (such as files), or impact the interpretation for previously executed operations (*e.g.*, *Undo/Redo* operations). Tracking the impact of process-level operations requires complex bookkeeping within RCF of current and previous states, to maintain an accurate characterization of the current design within the symbolic design model.

There are several possible semantic models for interpreting Undo/Redo operations. Our model (as dictated by the operational semantics of MicroStation95) limits the set of operations that are

Create	define a new object from scratch
Copy	define a new object by reference to a previously defined object
Delete	delete a previously created object
Modify	change structural aspect of a design object
Manipulate	reorient an object in space
Connect	create a joint between two objects
Import	read in a predefined part
Annotate	set/reset the semantic attributes of a design object
Analyze	invoke an evaluation tool for analyzing some aspect of the design

Table 1: Base-level Design Events

Undo	undo the previous 'undoable' operation
Redo	redo the last undone operation
File-Open	create or read-in a design file
File-Copy	copy a design file
File-Save	save a design file

Table 2: Process-level Design Events

'undoable'. In particular, only the base-level operations Create, Copy, Delete, Modify, Manipulate, and Annotate can be undone, and only Undo events can be Redone. A given Undo or Redo operation impacts the most recent operation for which the Undo or Redo is applicable that has not already been undone or redone.

Many tool events have direct correspondents in the design event model (*e.g.*, Create-Slab tool events map to a Create operation for objects of type Slab). However, certain design events correspond to a *set* of tool events (*e.g.*, joint creation consists of an origin definition followed by a joint declaration). At present, there are no *1:n* mappings within the framework, although macro definitions would require such a capability.

Several properties are maintained for each design event. The *type* property stores one of the base- or process-level event categories described above. The *tool events* property indicates the monitored operations that formed the basis for the given design event. The *status* property tracks whether the event is *:alive* or *:inactive*, relative to Undo/Redo operations. The *status* for an event starts out *:alive*, but will switch to *:inactive* should the event be 'undone' at some later stage. The *status* will be reset to *:active* should the undone event subsequently be redone. The role of the *focus object* property varies with the type of the design event. For example, with a Create event, the *focus object* corresponds to the created object; for a Delete event, the focus object corresponds

Construction	*Create, Copy, Import, Part, Connect*
Revision	*Undo, Redo, Delete, Modify, Manipulate,* *Create* (that have been undone but not redone), *Annotate* (that change rather than initiate values)
Deletion	*Delete, Undo*
Assembly	*Connect*

Table 3: Design Event Supertypes

to the deleted object. The *impact* property stores rationale-related information for this event.

Additional type-specific properties are stored with design events. For example, an Analysis event includes information about objects used for the analysis and their attributes, the analysis program invoked, and the results of the analysis. An Annotate event stores information about the attributes that are being modified, along with their newly assigned values. A Copy operation records the source object that was copied and the newly created object.

2.3.2 Design Event Supertypes

Design event types distinguish design operations with significant differences in their operational impact. Certain of the rationale generation techniques described later (Section 6.2) employ more abstract categorizations of designer activity, organized around the set of *design event supertypes* in Table 3.

2.3.3 Symbolic Design Model

The symbolic design model provides an abstracted representation of the emerging design that supports the reasoning required by the rationale construction process. It consists of representations for key elements within the design and relationships among them. It excludes certain information stored within the CAD model of the design (in particular, geometric information plays only a limited role), but augments the CAD representation to include relevant process information for objects within the design.

The representations employed support a rich set of design elements, spanning solid objects, features (structural attributes of an object, usually corresponding to a machining operation), two-dimensional elements used in the drafting process, and points. (Linear elements and points in and of themselves are not part of a design; rather, they are building blocks used to construct objects within the model.) Components can be defined in terms of other components and features; nested features are also supported.

Several categories of information are stored for a given design object. First, there is the standard definitional information: the object's geometric category (*i.e.*, sphere, slab, line) and key *structural attributes* (*i.e.*, the diameter for a sphere). In addition, the *semantic category* and category-specific *semantic attributes* are stored. Semantic attributes encapsulate properties related to the intended semantics of the object. The semantic attributes are determined by the *semantic category* of the object. For example, an object in the semantic category *gear* could have semantic attributes such as *number-of-teeth*, *ratio*, or *width*.[3] The specification of attributes can provide much insight into the evolution of a design. For this reason, the RCF system keeps a record of the evolving values for each attribute (both structural and semantic) that enables retrieval of values for any stage of the design process.

Several interobject relationships are stored for use in reasoning about rationale, including *parent/child* relationships that reflect hierarchical structuring of complex objects, *copies/source* relationships, and *attachment* relationships indicating connection of two design objects through a *joint* of a designated type. We define an *assembly* to be the closure of a set of objects under the attachment relationship (*i.e.*, under joint connectivity).

On the process side, several additional forms of information are stored, including a record of all operations performed on the object, analyses related to the object, time devoted to that object, the origins of the object (*i.e.*, selected from a part library, copied from a user, created by a designer), and status information (*:alive* or *:inactive*). In addition, a link connects each object back to the corresponding *tool object* in the CAD model.

3 Case Study: Surgical Robotic Arm

Arm-Inertia-I_X	$<$	*5 lb-in^2*	*Working Envelope*	within	*12 in*
Arm-Inertia-I_Z	$<$	*50 lb-in^2*	*Mass*	$<$	*2 lb.*
Inertia (unbalanced)	$<$	*1 lb-in^2*	*Pitch Servo Resonance*	$<$	*35 Hz*
Precision (unloaded)	$<$	*0.004 in*	*Pitch Cable Resonance*	$<$	*70 Hz*
Precision (loaded)	$<$	*0.01 in*	*Arm Mechanical Resonance*	$<$	*70 Hz*
Tool Friction	$<$	*0.75 lb-in*	*Tool Torque*	$<$	*16 lb-in*
Max. Stress to Yield Stress	$>$	*0.4*			

Table 4: Requirements for the Robotic Arm Design Case

[3] As described in Section 2.1, enhancements to the underlying CAD system support the explicit assignment of a semantic category and corresponding semantic attributes. A more advanced CAD system would provide such capabilities through the selection of semantically grounded components from predefined component libraries.

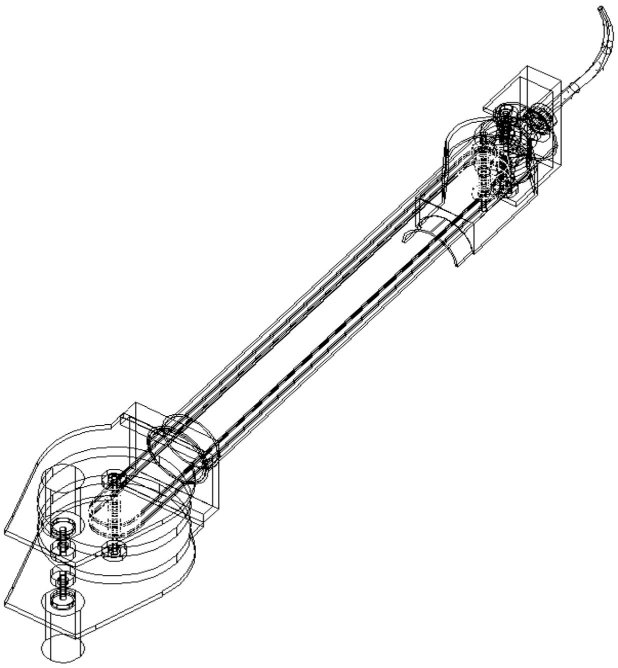

Figure 2: CAD Model for the Surgical Robot Arm

RCF was evaluated in a case study involving the design of a three-degree-of-freedom surgical robotic arm. The main technical challenges for this design were to provide sufficient actuation torque, while maintaining low inertia and high precision. Table 4 provides detailed information about the design requirements for this case.

RCF recorded designer activity over several versions of the arm, starting from a rough initial design, through various stages of refinements and optimizations. The designer divided the design into three main subassemblies: the *base assembly*, including the motors, the *arm assembly*, including the transmission, and the *wrist assembly*, including the end effector and tool. Figure 2 shows the resultant CAD model. From the recorded events, RCF was able to summarize designer activity at varying levels of abstraction, to identify phases where the designer concentrated on various parts or subassemblies or where design parameters where tuned, to track the results of design tradeoffs, and to explain key design changes. Examples from this case study will be used throughout the remainder of the paper to illustrate the workings of RCF.

4 Rationale Extraction: Overview of Technical Approach

We interpret design rationale broadly to include any aspect of a design session that could further understanding of the resultant design and the process by which it was developed. RCF provides two categories of rationale information: *session content* and *designer intent*. Session content focuses on summaries and abstracted views of the design process, organized from several different perspectives. These perspectives support different approaches to comprehending the design, each suited to a different set of needs. Rationale related to *designer intent* provides explanations for key design changes. Rationale extraction is organized around a set of domain-independent *design metaphors* (Section 4.2) augmented by limited amounts of task- and domain-specific design knowledge (Section 4.3).

4.1 Rationale Categories

4.1.1 Session Content

Comprehension of the evolution of a design requires records of both complete histories of designer activities and snapshots of the current and past states. Toward this end, *session content* focuses on characterizations at multiple abstraction levels of events and objects for a given design session. At the lowest level, complete and detailed descriptions of all design events and objects are provided. Above that, a variety of views at higher levels of abstraction aggregate events and design objects into related units. These abstracted summaries provide broad overviews of the design and its evolution.

RCF provides summaries of a design session from three different perspectives. The *effort-centered* perspective summarizes where and how a designer spent his or her time (Section 5). The *event-centered* perspective summarizes the design session at multiple abstraction levels, using a combination of design metaphors and clustering methods to perform the abstractions (Section 6). The *object-centered* perspective provides historical and explanatory information for individual design objects and groups of objects that may be explicit in the design (*e.g.*, assemblies) or inferred to be related by RCF (Section 7).

4.1.2 Design Intent

A finished CAD model shows the end product of a designer's efforts but omits the *changes* that were made in the development of the design. Changes provide insight into the evolution of the design, showing alternative paths that the user explored and basic strategies used to produce the final results. For this reason, a key focus within RCF has been to identify and explain changes to a design, with emphasis on the following capabilities:

- Explain the motivations for changes in the design.
- Aggregate operations with a common purpose into coherent sets.
- Summarize considered options with possible explanations for their acceptance or rejection.

In the terminology of (Gero, 1990), we focus on the *evaluation* and *reformulation* of the design: the designer compares the desired behavior of the ideal artifact with the actual behavior of the proposed structure. Where the comparison is unsatisfactory, the designer attempts to modify the structure accordingly. We assume that the desired behavior can be expressed as a set of design requirements, and that the analyses used to perform the evaluation are available for inspection by RCF.

We have explored an approach that involves situating changes within contexts defined by *clustering* related events, and reasoning with *domain knowledge* about qualitative effects of design operations. This work is described in Section 8.

4.2 Design Metaphors

Design metaphors are multistep patterns of events (not necessarily contiguous) that describe episodes of coherent designer activity. They can be applied at varying scales of resolution: at the design event level, or over groupings or abstractions of design events. RCF contains a suite of design metaphors whose recognition enables generation of defeasible explanations of a range of designer activity. Two example metaphors are presented here.

- The *Refinement* metaphor consists of a cycle of *Analyze X - Revise* behavior, indicating that the designer is focusing on a particular design requirement *X*. It is reasonable to infer that intervening modifications to the design are performed with the goal of addressing *X*, although not all such revisions will have been performed with *X* in mind. Thus, while the metaphor does not definitively link action and intent, it provides a plausible explanation for the designer's actions.
- The *1:1 Part Substitution* metaphor captures the notion that the designer has swapped one functional component for another. In particular, Part *B* is considered to be substituted for

Part *A* when it is observed that first, Part *A* is removed, and then some Part *B* from the same functional category is added to the assembly with the same connectivity as Part *A*. These part operations must occur within a certain window of activity – namely, an interval of operations that the system has identified as being related (see Section 6.3) – but need not be consecutive.[4]

The *Refinement* and *1:1 Part Substitution* metaphors capture general design principles and as such are applicable to a broad range of design tasks. To date, all design metaphors within RCF share this domain-independence. Task-specific design metaphors could readily be added to the system to increase explanatory power, although at the cost of the knowledge engineering involved. As an example, consider the choice of a power regulator for a control system. Suppose that the designer chooses a DC-DC converter to step down from a 12V power supply to the required 5V control supply, rather than a shunt regulator, the standard alternative. DC-DC converters are, in general, more expensive and less reliable, but more efficient than shunt regulators. So, in the context of supplying power to a circuit, it is reasonable to infer that the choice of DC-DC converters indicates that power efficiency was a primary goal of the designer. Here, a fairly strong inference can be made. In general, task-specific metaphors support correspondingly deeper explanations of designer actions; however, that increased detail comes at the cost of the knowledge engineering required to encode metaphors for the given domain.

4.3 Domain Knowledge

The use of background knowledge can greatly extend the rationale extraction capabilities. However, such knowledge can be difficult and expensive to acquire and represent. For this reason, we explored a range of techniques that vary in the amount of background knowledge that they employ. RCF employs two kinds of optional background knowledge, which are used primarily to support explanation of certain forms of designer activity (as described in Section 8).

- Overall *design requirements* are represented as a collection of properties, possibly with threshold constraints that must be satisfied (*e.g.*, $\textit{Arm-Inertia-}I_z < 50\ \textit{lb-in}^2$). Nonmeasurable requirements, such as *Durability*, do not include explicit thresholds. Design requirements are used within RCF to provide possible motivations for designer activities.

[4] Additional substitution metaphors could be defined that capture, for example, one-to-many and many-to-one substitutions.

- *Qualitative models* of the effects of design operations provide linkage from observed activities to their impact on key design properties. Within RCF, these models enable casually grounded explanations of designer activity.

5 Effort Allocation

The manner in which a designer allocates his or her efforts can provide valuable insight into the design. Time spent, level of detail, and the number of alternatives explored can all indicate what the designer thinks is important or what has proven to be difficult within a given design task.

RCF tracks effort with respect to *operation allocation* and *time allocation*. Operation allocation is measured in terms of number of design events. Time allocation is measured in terms of clock time devoted to design events. In particular, tool events are tagged with normalized timestamps to provide a basis for our temporal allocation model. The duration between timestamps is 'wall-clock' time, including both the time required to perform the operation, and inactive time (*i.e.*, between tool operations). A normalization process eliminates large time gaps between tool commands, which are assumed to correspond to designer idle time; tool events that occur after such gaps are assigned a default duration to generate the next timestamp. The time for a design event is defined to be the difference between the normalized timestamps for the corresponding tool event and the tool event for the previous design event.

Time and operation allocation can be reported relative to a given design object or a class of design objects. In addition, time allocation can be obtained for a given design event or a class of design events. Relative distributions across design event types and objects are also supported.

These effort allocation models have certain drawbacks. Characterizing operational effort in terms of number of design events ignores the relative significance of different kinds of operations. For example, an operation that reads in a previously defined complex part would contribute only a single increase to the operation tally, while recreation of the part by hand could require a substantial number of operations. In certain situations, a weighted model for operations might be appropriate; however, effort measured in terms of operation count still provides insight into where the designer focused his or her effort.

Issues arise as well with the model for time allocation. A 'credit assignment' problem arises in determining time allocation in that several preparatory operations may be required for a given creation operation. For example, consider the creation of a linear profile with a complex shape that is to be used (with minor modifications) as the basis for a number of free-form solids. Ideally, time allocated to the creation of the profile should be distributed over all shapes built with that profile. The complexity of the problem increases once modifications and revisions are taken into account.

Despite the crudeness of the underlying models, effort allocation information can provide useful insights into understanding a design. For instance, when faced with the task of modifying a design, knowledge that the original designer spent a significant amount of effort on a specific component should alert the downstream designer that subtle design issues may be at hand. Alternatively, information that a component received little attention could indicate either that the component is relatively unconstrained (and hence was easy to design), or that the designer simply lacked sufficient time to refine it.

6 The Event-centered Perspective

The event-centered perspective provides summaries of a design session at varying levels of abstraction. To support these different views, RCF first partitions the design event log into *versions* of artifacts (Section 6.1). Within a given version, RCF then aggregates designer activity into higher-level abstractions. Individual design events are grouped into *part-level operations* (Section 6.2), which focus on design objects at the level of parts in an assembly. Next, part-level operations are grouped into *activity phases* (Section 6.3), which correspond to broader collections of activities with a common design objective.

6.1 Versions

Versioning relies on the observation of coherent sets of operations within a design file. Within RCF currently, the metaphor *Create/Copy File - Activity - Save File* defines version boundaries; (Qureshi et al., 1997) employs a similar convention. Clearly, more general metaphors are possible (as discussed further in Section 10).

Within a version, the system tracks the different activity phases of the designer, the subsystems of the design worked on the most, the design requirements that were addressed, and significant structural and attribute changes. Figure 3 summarizes designer activity for one version (Version 2) of our design case study. (More detailed information about the *activity phases* within this summary is provided in Section 6.3.) Figure 4 summarizes the changes between Versions 1 and 2 of this same design.

```
***************** Version 2 *********************

 From event 218 to 374

>>>>>>>>>>>>>>>>> Activity Phase 0 <<<<<<<<<<<<<<<<
        Activity type: CONSTRUCTION
Primary focus on (ARM3 ARM)

>>>>>>>>>>>>>>>>> Activity Phase 1 <<<<<<<<<<<<<<<<
        Activity type: REVISION
In the BEGINNING, no detected focus
In the MIDDLE, MILD focus on (WRIST)
In the END, STRONG focus on (BASE)

>>>>>>>>>>>>>>>>> Activity Phase 2 <<<<<<<<<<<<<<<<
        Activity type: REFINEMENT
In the BEGINNING, focus on (ARM-INERTIA-IZ WEIGHT)
     Refinement of ARM-INERTIA-IZ:
           (60.0 50.0) --> REDUCE
   Modifications primarily on
     DOBJ92 ( BASE_GEAR ) from assembly BASE
     DOBJ207 ( BV_1250 ) from assembly WRIST

     Refinement of WEIGHT:
           (2.2 2.0) --> REDUCE
   Modifications primarily on
     DOBJ92 ( BASE_GEAR ) from assembly BASE
     DOBJ207 ( BV_1250 ) from assembly WRIST

In the END, focus on (STRESS)
     Refinement of STRESS:
           (0.6 0.4) --> REDUCE
   Modifications primarily on
     DOBJ31 ( BV_0625 ) from assembly WRIST
     DOBJ207 ( BV_1250 ) from assembly WRIST

>>>>>>>>>>>>>>>>> Activity Phase 3 <<<<<<<<<<<<<<<<
        Activity type: CONSTRUCTION
In the BEGINNING, STRONG focus on (GROUP4 BEAR_BR)
In the MIDDLE, STRONG focus on (GROUP4 BEAR_BR)
In the END, MILD focus on (GROUP4)

>>>>>>>>>>>>>>>>> Activity Phase 4 <<<<<<<<<<<<<<<<
        Activity type: ANALYSIS

From event 371 to event 374
Analyses performed:
  Analysis PRECISION-LOADED, with outcome 0.1; design requirement UNSAT
  Analysis PRECISION-UNLOADED, with outcome 0.018; design requirement UNSAT
  Analysis SERVO-RES, with outcome 50.0; design requirement UNSAT
  Analysis FRICTION, with outcome 2.0; design requirement UNSAT
```

Figure 3: Summary of Activity within a Single Version of an Artifact

```
>>> Summary of Part Changes between Version 1 and Version 2 <<<

Deletions:
  Part GR_16_1 : all instances deleted
  Part GR_08 : all instances deleted
  Part BV_1875 : all instances deleted
  Part BASE_PULLEY1 : all instances deleted

Additions:
  Part ARM4 was added
  Part BASE_PULLEY_3 was added
  Part BEAR_BR was added
  Part BELT was added
  Part BRNG_W_S3 was added
  Part BV_1250 was added
  Part GEAR_16_3 was added
  Part GR_08_4 was added
  Part TOOL_BRNG was added

Modifications:
  Part BASE_GEAR was modified
```

Figure 4: Summary of Part Changes between Two Versions of a Design

6.2 Part-level Operations

Part-level operations aggregate individual design events into higher-level units that focus on design objects at the level of parts in the assembly. Thus, rather than examining activity on the level of features or components being modified or joined, a part-level chronology consists of parts being created, added or removed from the assembly, substituted for other parts, or modified. The part-level view of the design process is both more understandable to human observers and more convenient for recognizing abstract design metaphors.

Part operations, shown in Table 5, are characterized in terms of the event supertypes displayed in Table 3. By necessity, the part-level models within RCF are somewhat implementation-specific to MicroStation95.

Figure 5 shows an example part-level abstraction produced by RCF. The excerpt from the design event log (on the left) constitutes a period of revision activity, in which the designer replaces selected components in the design. Within MicroStation95, a *joint* connects two design objects at a contact point; disconnecting a component from an assembly generally requires a series of joint deletions. RCF maps a joint back to the components that the joint connects, and thus keeps track of parts being switched in and out of the design. The part-level description on the right abstracts the explicit joint manipulations into a summary of the parts being removed, added, or replaced. Each object within RCF has a unique *design object id*, for example, `DOBJ167`. Reference to a design object corresponding to an instance of a part generally includes the name of the original part from

Part Creation	A series of construction and revision events on a set of related design objects, culminating in a part-declaration command, defining a representative component of this set of objects as a part.
Part Addition	Assembly and construction events for objects that are parts, design objects derived from parts, or joints between parts. Operations on features and subcomponents are explicitly excluded, corresponding instead to *part modification* events on the parent component of the feature.
Part Removal	Events that delete a part or all joints that link the part to other parts within an assembly.
Part Modification	Part-level revision events and feature-level creation or revision events on children of a part component.
Part Substitution	Defined by the 1:1 Part Substitution metaphor (described in Section 4.2).

Table 5: Example Part Operations

```
Devent 263: DELETE JOINT90
Devent 264: DELETE JOINT91
Devent 265: CONNECT JOINT92
Devent 266: CONNECT JOINT93
Devent 267: DELETE JOINT65
Devent 268: DELETE JOINT66
Devent 269: CONNECT JOINT94
Devent 270: CONNECT JOINT95
Devent 271: DELETE JOINT86
Devent 272: DELETE JOINT87
Devent 273: CONNECT JOINT96
Devent 274: CONNECT JOINT97
Devent 275: DELETE JOINT80
Devent 276: CONNECT JOINT98
Devent 277: DELETE JOINT81
Devent 279: UNDO
Devent 280: DELETE JOINT82
```

```
From design event 263 to design event 280
   Assemblies involved:
     MAIN-ASSEMBLY
     WRIST
     ARM
   Parts added:
     DOBJ229 (GEAR_16_3) from assembly WRIST
     DOBJ228 (GEAR_16_3) from assembly WRIST
   Parts removed:
     DOBJ183 (GR_08) from assembly WRIST
   Part substitutions:
     DOBJ167 (BV_1875) was replaced by DOBJ227 (BV_1250)
     DOBJ168 (BV_1875) was replaced by DOBJ226 (BV_1250)
     DOBJ212 (ARM3) was replaced by DOBJ225 (ARM4)
     DOBJ213 (ARM3) was replaced by DOBJ224 (ARM4)
```

Figure 5: Abstracting from Design Events to Part-level Operations

which it was created. Thus, `DOBJ212` and `DOBJ213` are both instances of the previously created part `ARM3`.

6.3 Activity Phases

Activity phases are groupings of events that describe the designer activity at the level of abstract operations on components, parts, or the design artifact itself. Four types of activity phases are extracted from the sequence of part operations; these types are summarized in Table 6. Analysis and part-level statistics are kept for each type.

During a given activity phase, a designer will often focus on a specific part or set of parts

Construction	a period of interleaved part creation and part addition events
Revision	a period of interleaved part revision, part addition, part deletion, and/or part substitution
Analysis	a period of analysis events not linked to any revisions.
Refinement	a period of analysis events and related revision events

Table 6: Types of Activity Phases

```
>>>>>>>>>>>>>>>>> Activity Phase 1 <<<<<<<<<<<<<<<<
        Activity type: REVISION
In the BEGINNING, no detected focus
In the MIDDLE, MILD focus on (WRIST)
In the END, STRONG focus on (BASE)

>>>>>>>>>>>>>>>>> Activity Phase 2 <<<<<<<<<<<<<<<<
        Activity type: REFINEMENT
In the BEGINNING, focus on (ARM-INERTIA-IZ WEIGHT)
     Refinement of ARM-INERTIA-IZ:
           (60.0 50.0) --> REDUCE
   Modifications primarily on
     DOBJ92 ( BASE_GEAR ) from assembly BASE
     DOBJ207 ( BV_1250 ) from assembly WRIST

     Refinement of WEIGHT:
           (2.2 2.0) --> REDUCE
   Modifications primarily on
     DOBJ92 ( BASE_GEAR ) from assembly BASE
     DOBJ207 ( BV_1250 ) from assembly WRIST

In the END, focus on (STRESS)
     Refinement of STRESS:
           (0.6 0.4) --> REDUCE
   Modifications primarily on
     DOBJ31 ( BV_0625 ) from assembly WRIST
     DOBJ207 ( BV_1250 ) from assembly WRIST

>>>>>>>>>>>>>>>>> Activity Phase 3 <<<<<<<<<<<<<<<<
        Activity type: CONSTRUCTION
In the BEGINNING, STRONG focus on (GROUP4 BEAR_BR)
In the MIDDLE, STRONG focus on (GROUP4 BEAR_BR)
In the END, MILD focus on (GROUP4)
```

Figure 6: Example Activity Phases

for some time before switching attention to another aspect of the design. Using the effort allocation models described in Section 5, RCF identifies the evolving focus of designer attention, at the level of design requirements being addressed, individual parts, subassemblies of parts, and *implicit groups* of parts that are identified automatically during the extraction of activity phases (see Section 7.2).

The *focus* during an interval of activity is defined as a part, a grouping of parts, or a design requirement, for which the percentage of effort devoted to it exceeds some threshold. It is possible for a period of activity to have no detected focus. The effort metric used to determine attention focus can be either number of operations or accumulated time per part. For simplicity, RCF simply subdivides an activity phase into three equal subintervals (`BEGINNING`, `MIDDLE`, and `END`). More generally, the number and length of subintervals could be computed dynamically by applying clustering techniques to define focus areas.

Figure 6 shows example revision and construction types of activity phases and the detected foci within each. In Activity Phase 3, note the focus on an implicit group of parts, `group4`, a set of bearings. The parts in `group4` are not explicitly linked in the design, nor are they even members of the same subassembly; they were grouped because activity on any one part in the group correlated strongly with activity on other parts in the group. Section 7.2 discusses the methods used to identify such *implicit groups*.

7 The Object-centered Perspective

The object-centered perspective provides information about the design process with respect to a particular design object, or set of objects. Two types of object-centered information are produced by RCF. Explicit *object histories* are maintained for individual and aggregated objects. In addition, RCF looks for interesting relationships among parts or between a part and design requirements, based on designer activity.

7.1 Object Histories

For each design object, RCF maintains a detailed history that includes the object's origins (how and when created), related design events, connections to other objects, and effort expended for that object. Additional information is kept for parts: related part-level operations, a detailed part history, and *pedigree* information. Pedigree information refers to the heritage of a part. For example, a part can be selected from a library or a catalog, it can be created from scratch, or it can be a

```
----------  Part History: ARM4  ----------

 --- Source ---
 Design object DOBJ214
  is Part ARM4 of type ARM
  from file c:/models/version1/arm4.dgn
  and is originally derived from part ARM3

 --- Part Activity ---
 In Version 2, Activity Phase 1, type REVISION:
  >> added to assembly as replacement for ARM3


----------  Version Info:  Part ARM4  ----------

Part ARM4 was originally copied from Part ARM3

Part ARM3 was originally copied from Part ARM2
    WIDTH changed from 0.0 in ARM2 to 1.6 in ARM3
    HEIGHT changed from 1.6 in ARM2 to 0.0 in ARM3

Part ARM2 was originally copied from Part ARM1
    WIDTH changed from 2.25 in ARM1 to 0.0 in ARM2
    HEIGHT changed from 2.5 in ARM1 to 1.6 in ARM2
```

```
----------  Part History: BV_1250  ----------

 --- Source ---
 Design object DOBJ207
  is standard part BV_1250 of type GEAR
  from file c:/models/version1/std_1250.dgn

 --- Constraints ---
 BV_1250 (GEAR), attributes (MATERIAL)
   constrains
 BV_0625 (GEAR), attributes (MATERIAL)

 --- Part Activity ---
 In version 2, activity phase 1, type REVISION:
  >> Added to assembly as replacement for BV_1875

 In version 2, activity phase 2, type REFINEMENT:
 >> Modified
     Event 305: ANNOTATE CAT_NUM ==> BERG_M48N2A;
       --> Possible Effects: (ARM-INERTIA-IZ DOWN)
                             (WEIGHT DOWN)

 >> Modified
     Event 313: ANNOTATE CAT_NUM ==> BERG_M48N2A;
                         MATERIAL ==> SS;
       --> Possible Effects: (WEIGHT UP)
                             (STRESS DOWN)
```

Figure 7: Example Part Histories and Part Version Summaries

copied from another part. A part can have several *versions*: for example, if *PartB* is created by making a copy of *PartA*, and then modifying it, then *PartA* is considered to be a previous version of *PartB*.

A part's history includes when it was introduced into the assembly, if and when it was removed, whether it was replaced by another part (*i.e.*, instances of the *1:1 Substitution* metaphor), and modifications to the part or any of its attributes. The context of these operations is reported: which version and what type of activity phase. Also recorded are hypothesized explanations for activities related to the part (see Section 8), detected relationships between a part and any design requirements, and inferred dependencies on other parts through membership in *implicit groups*.

Figure 7 shows the part histories for parts `ARM4` and `BV_1250`. `ARM4` was copied from `ARM3`, and then replaced it in the assembly. `ARM3` was copied from `ARM2`, and then modified. The modifications made at the time of the part copy are recorded and shown. `BV_1250` was added to the assembly as the replacement for another part. Furthermore, RCF hypothesized an implicit constraint between the `material` attributes of `BV_1250` and `BV_0625`.

Insertion Group	Parts that are consistently added together to the assembly during the same period of construction activity (not necessarily in the same order). (Metaphor: *Insert* A_0 ... *Insert* A_n *(repeated)*)
Modification Group	Parts that are consistently modified together during the same period of part revision (not necessarily in the same order). (Metaphor: *Modify* A_0 ... *Modify* A_n *(repeated)*).
Functional Group	Parts that are consistently refined together within a single refinement phase, in connection with the same design requirement. (Metaphor: *Analyze X - Modify* A_0 ... *Modify* A_n *- Analyze X (repeated)*).

Table 7: Example Implicit Groups

```
 >>>>>>>>> Implicit Groups <<<<<<<<<<
Group INSGP0 of type INSERTION: parts MOTOR TOOL TOOL_BCK1
Group INSGP1 of type INSERTION: parts PINIONN PINION
Group INSGP2 of type INSERTION: parts GEAR_16_3 BV_1250 ARM4
Group INSGP3 of type INSERTION: parts GR_08_4 GR_08_3
Group INSGP4 of type INSERTION: parts TOOL_BRNG BRNG_W_S3 BEAR_BR
Group INSGP5 of type INSERTION: parts B_SPOOL_1 W_SPOOL_1 CABLE
Group MODGP6 of type MODIFICATION: parts WIST_PULLEY_1 WRIST_SHAFT
Group MODGP7 of type MODIFICATION: parts PINION PULLEY1_B
Group MODGP8 of type MODIFICATION: parts GR_16 GR_08 ARM2
Group MODGP9 of type MODIFICATION: parts BASE_PULLEY_3 GEAR_16_3 GR_08_3 GEAR_08_V3 ARM3
Group MODGP10 of type MODIFICATION: parts BRNG_W_S3 BRNG_WRIST_S2 BEAR_W
Group MODGP11 of type MODIFICATION: parts CABLE B_SPOOL_1 W_SPOOL_1
```

Figure 8: Implicit Groups Detected during a Design Session

7.2 Object Abstractions: Implicit Groups

The aggregation of objects into logically related groups is a powerful mechanism for improving the understandability of complex structures. Assemblies and hierarchies provide examples of groups that a designer defines explicitly. In addition, ‘hidden’ relationships can be present in a design that, if made apparent, could similarly improve understanding. For example, two parts (possibly from different subassemblies) may be implicitly dependent on each other, either structurally or functionally, in a way that would not be apparent from examination of the finished CAD model.

RCF searches for *implicit groups* of design objects that satisfy such hidden relationships. While the system may or may not be able to identify precise constraints among the parts in an implicit group, it can bring them to the attention of a designer, who may be able to identify a reason for such a dependency. RCF contains design metaphors for recognizing several types of implicit groups, which are listed in Table 7. Figure 8 summarizes several implicit groups detected during a portion of the robotic arm design session.

The period of time over which one looks for these part activity patterns impacts the set of groups that are extracted. One could look for patterns conservatively, over the entire session. This strategy would result in groups that are likely to be significant; on the other hand, other significant patterns, which may exist only for a shorter interval of the session, will be missed. These groups would be discovered by more localized search, over a single version of the design, for example. In this case, however, one might extract spurious, coincidental patterns of activity. Currently, functional groups are local to a single refinement phase, and are taken with reference to the associated analysis. Other implicit groups are taken over the whole session.

8 Explaining Changes

The linking of activity with intent constitutes a key component of rationale. RCF reasons about designer intent during refinement phases, using a set of methods that vary in the amount of background knowledge about the design task that they require.

The approach involves gathering *evidence* to support a range of *hypotheses* as to the designer's motivation for performing particular operations or groups of operations. The hypotheses postulate that the designer intended to impact some design property (*e.g.*, *Torque*), possibly to move in some specified direction (*e.g.*, *increase* or *decrease*). The evidence is based on three sources of information: observed operations by the designer, design requirements, and information about the impact that various changes could have on design requirements. As discussed further below, the latter two types of evidence are not required, but enable richer explanations when provided. A calculus for combining evidence provides inference of defeasible conclusions about designer intent.

8.1 Clusters, Hypotheses, and Evidence

To provide a context in which to relativize explanations, related change events are grouped into *change clusters*. Analysis events provide the basis for extracting change clusters, in accord with the metaphor $Analysis_1 \ldots\ Analysis_k - Event_1\ \ldots\ Event_j - Analysis_{k+1}\ \ldots\ Analysis_n$. The design properties tested by the analyses within the cluster are adopted as hypotheses for explaining the motivation for changes performed within the cluster. For example, during the development of the robotic arm, the designer may conduct an analysis to determine the resonance of the arm, perform some additional modifications, calculate the arm inertia, recalculate the resonance, and then perform modifications to a different part of the design. In this case, the system would extract a cluster that begins with the first resonance analysis and ends with the second resonance analysis.

The hypotheses for this cluster would be that the designer is trying to impact resonance and inertia.

RCF collects *evidence* to strengthen or weaken these initial hypotheses. Table 8 displays the types of evidence currently employed within the system. `MATCHED-ANALYSES` correspond to the inclusion of a pair of analyses, one from the initial set of analyses and one from the final set, that analyze the same design property. For example, the resonance analyses in the above example satisfy this criterion. As opposed to observing analyses in isolation, matched analyses serve as an indicator that (at least some of) the intervening operations were likely performed to impact the design property that the analyses evaluate. Detection of a `REFINEMENT-TREND`, whereby an analyzed design property changes monotonically within a cluster, would similarly increase the likelihood that the cluster is focused on impacting that design property. `MATCHED-ANALYSES` and `REFINEMENT-TREND` evidence are extractable directly from observed design events. The remaining evidence types require additional background knowledge related to design requirements and effects of design operations; they are discussed in Section 8.2.

Type of Evidence	Description	Knowledge Requirements
MATCHED–ANALYSES	The initial and final sets of analyses in the cluster both include an analysis that evaluates the design property.	None
REFINEMENT–TREND	The cluster contains a monotonic trend (of length greater than two) in the value of the design property.	None
PREVIOUSLY–COUNTERED	Modifications were made in a preceding cluster that moved the design property in the wrong direction with respect to its associated design requirements.	*Design Requirements*
KNOWN–SAT KNOWN–UNSAT	Requirements for the design property are known to be satisfied (unsatisfied) at the beginning of the change cluster.	*Design Requirements*
MODIFY	An event that could possibly impact this design property occurred in the cluster.	*Effects*
MODIFY–WRONG–DIR MODIFY–RIGHT–DIR	Modifications were made that moved the design property in the wrong (right) direction with respect to its associated design requirements.	*Design Requirements, Effects*

Table 8: Types of Evidence to Support Hypotheses for Impacting a Design Property

An *evidence calculus* provides the basis for combining collected evidence to produce final interpretations. Hypotheses that score over a designated threshold are considered as intended effects; as such, we do not assume that a designer has a single objective within a change cluster.

Different viewpoints or perspectives on the design process can be supported by assigning different levels of significance to various types of evidence. RCF distinguishes two categories of evidence: *cumulative* and *primary*. Cumulative evidence simply increases or decreases the likelihood of a hypothesis in a straightforward manner. Primary evidence is deemed *necessary* for a conclusion to be drawn, thus providing the means to filter irrelevant observations. This technique allows the observer to specify the level of granularity that the system should use to draw conclusions. For example, the observer may want a coarse level of granularity, only considering the designer's intentions over an extended period of time. In this case, he or she may decide to designate `REFINEMENT-TREND` evidence as necessary in order for an effect to be considered possibly intended. Conversely, if an observer wishes to examine the design process on a finer scale, considering all possible effects on an event-by-event basis, he or she may choose not to designate any evidence as primary. RCF interpretations also support specifications of weights for emphasizing different types of evidence.

Currently, RCF provides two interpretations. A *fine-grain* interpretation considers all observed effects, with no primary evidence. A coarser *context-emphasizing* interpretation requires `MATCHED-ANALYSES` evidence for an effect to be considered as intended. Both interpretations combine cumulative evidence additively, although this need not be the case. In a probablistic model, for example, evidence would be combined multiplicatively.

8.2 Background Knowledge

The evidence types described above are grounded in directly observable designer activity, and as such can be applied to a wide range of design tasks. However, they have limited explanatory power. Substantially more interesting explanations of designer intent can be generated through the use of background knowledge. We consider two types: task-specific *design requirements* and general-purpose domain knowledge about the *effects* of operations.

Design Requirements

When available, quantitative knowledge about task-specific *thresholded design requirements* (*i.e.*, maximum or minimum thresholds) can be used to further weaken or strengthen belief in hypotheses. RCF employs five types of evidence related to design requirements. If a change occurs that increases the degree of satisfaction of a design requirement that is already known to be satisfied, it is less likely (but not impossible) that the designer was intentionally focusing on that design requirement (*i.e.*, `KNOWN-SAT` from Table 8). Similarly, knowing that the requirement is not satisfied would increase the likelihood that the designer was focusing on that requirement (*i.e.*, `KNOWN-UNSAT`). The evidence type `PREVIOUSLY-COUNTERED` indicates that modifications were made in one or more preceding clusters that moved the design property away from the direction required to satisfy stated design requirements (*e.g.*, as a side effect of addressing some other requirement). This increases the likelihood that the designer would be attempting to address this recently countered requirement. The evidence types `MODIFY-RIGHT-DIR` and `MODIFY-WRONG-DIR` require information about design requirements but also of effects; for that reason, they are discussed in the following section.

Effects of Changes: QIC Tables

Background knowledge of the possible *effects* of designer changes can similarly enable deeper explanations of designer intent than is possible with observed designer activity alone. We have developed an approach based on reasoning qualitatively about these possible effects. At the heart of the method lies the specification of a table of *qualitative impact of changes*, which encodes the effects that a given design change can have on designated design properties. This information can also be used to assign intended effects to specific events and to identify side effects not manifested by the analyses performed.

Figure 9 presents an excerpt from the *qualitative impact of change* (QIC) table for the robotic arm design case. Here, the focus is limited to changes to semantic attributes. However, the approach can be readily extended to handle structural changes (*e.g.*, the addition or deletion of objects) and magnitudes of changes. For a given change to a semantic attribute of an object, the QIC documents design-related properties that are affected, along with a specification of how those properties are impacted. As such, they can be viewed as possible *explanations* for performing a given design change. Attributes have one of three types. *Numeric* attributes show positive or negative correlation with design properties. *Binary* attributes show correlation when the attribute assumes values of `true` or `false`. *Enumerated* attributes specify correlation for various ranking functions.

QIC tables share with the *confluences* of (De Kleer and Brown, 1984) the goal of representing the qualitative effects of change. Confluences are equations defined in terms of qualitative deriva-

OBJECT CLASS	ATTRIBUTE	TYPE	RANK	Precision Unloaded	Precision Loaded	Torque Avail	Inertia Arm	Friction	Max Stress	Cost	Durability	Resonance Mech.	Resonance Servo	Mass
Gears	Ratio	NUMERIC		+		+				+			+	+
	Radius	NUMERIC			+		+		+	+			-	+
	Diametral Pitch	NUMERIC		+					+	+				
	Quality	NUMERIC		+				-		+				
	Width	NUMERIC					+		+		+		-	+
	Ct-Ct Tolerance	NUMERIC		+				-		+				
	Adjustable Ct-Ct Dist	BINARY	TRUE	+						+				
	Material	ENUM	Lighter				-						+	-
			Yield						+	+	+			
			Stress						+		+			

Figure 9: Excerpt from the QIC Table for the Robotic Arm Design Case

tives of variables whose values range over some limited domain (usually +, -, or 0). For example, the confluence $\partial P + \partial A - \partial Q = 0$ describes the behavior of a valve with flow Q, flow area A, and pressure P; here, ∂Q, ∂A, and ∂Q denote changes in Q, A, and P. While confluences can be used to represent relations among a range of changes to a device, QIC tables define only pairwise interactions between individual design changes and designated design properties. Within the scope of our work, QIC tables were defined by hand. However, techniques for reasoning about causal effects, such as propagation of disturbances among confluences (De Kleer and Brown, 1984) or causal ordering and comparative statics (Iwasaki and Simon, 1986; Iwasaki and Simon, 1994), could be used to derive QIC entries from formal descriptions of component behaviors.

Without QIC information, hypotheses are limited to the design properties tested by analyses within the cluster. When QIC information is available, the set of hypotheses is expanded to include the effects of all actions performed within the cluster. QIC information also provides the basis for two additional types of evidence. `MODIFY` evidence indicates that an action was performed that could possibly impact the design property in question. `MODIFY-RIGHT-DIR` and `MODIFY-WRONG-DIR` indicate that the designer made changes that moved the design property in the right/wrong direction relative to the stated design requirements.

Figure 10 presents summaries of two example clusters extracted by RCF. Each summary includes (a) the key design events for the cluster (here, analyses and changes to semantic attributes of objects), and (b) classification of effects of change operations within the cluster (extracted from the QIC table) as *intended* or *side* effects. Evidence to support the classification is provided, along with scores in the range $[-5, 5]$ determined by the application of the evidence calculus in use (here, the *context-emphasizing* calculus). A positive score denotes 'intended' while a negative score denotes 'unintended', with larger absolute scores indicating greater belief in the hypothesis.

Within these clusters, the change events consist of modifications to the semantic attributes of

```
>>> Cluster #1 <<<
Key Design Events:
 #306: ANALYSIS of ARM-INERTIA-IZ with result 60.0
 #308: ANALYSIS of WEIGHT with result 2.2
 #310: ANNOTATE DOBJ209 (BV_1250)
         CAT_NUM --> BERG_M48N2A;
 #312: ANNOTATE DOBJ92 (BASE_GEAR)
         CAT_NUM --> F32-A6-96;
         MATERIAL --> DELRIN (was ALUMINUM);
 #313: ANALYSIS of ARM-INERTIA-IZ with result 50.0
 #314: ANALYSIS of WEIGHT with result 2.0

 Intended Effects:
  ==> (WEIGHT DOWN) Likelihood = 2
     Evidence:
       (MATCHED-ANALYSES #308 #314)
       (MODIFY #312)
       (KNOWN-UNSAT #308)

  ==> (ARM-INERTIA-IZ DOWN)  Likelihood = 1
     Evidence:
       (MATCHED-ANALYSES #306 #313)
       (KNOWN-UNSAT #306)

 Side Effects:
  ==> (STRESS UP)  Likelihood = -1
     Evidence:
       (MODIFY-WRONG-DIR #312)
```

```
>>> Cluster #2  <<<
Key Design Events:
 #316: ANALYSIS of STRESS with result 0.6
 #318: ANNOTATE DOBJ209 (BV_1250)
         CAT_NUM --> BERG_M48N2A;
         MATERIAL --> SS (was ALUMINUM);
 #321: ANNOTATE DOBJ31 (BV_0625)
         CAT_NUM --> BERG_M48N3;
         MATERIAL --> SS (was ALUMINUM);
 #323: ANALYSIS of STRESS with result 0.4

 Intended Effects:
  ==> (STRESS DOWN)  Likelihood = 3
     Evidence:
       (MATCHED-ANALYSES #316 #323)
       (MODIFY #318 #321)
       (PREVIOUSLY-COUNTERED Cluster-1)
       (KNOWN-UNSAT #316)

 Side Effects:
  ==> (WEIGHT UP)  Likelihood = -1
     Evidence:
       (MODIFY-WRONG-DIR #318 #321)
```

Figure 10: Sample Change Clusters with Evidence

previously defined objects (*via* `ANNOTATE` design events). The two attributes changed are the material and catalog number (`CAT-NUM`).[5] In the first cluster, QIC knowledge is available only for the change in material for `DOBJ92` from `Delrin` to lighter `Aluminum`. RCF uses it to generate the hypotheses that the intent of the cluster is to reduce `weight`, decrease `inertia`, or increase `stress`. Based on additional evidence (such as the fact that the weight requirements were not satisfied at that point in the design), RCF's evidential reasoning calculus infers that Event 312 was performed to reduce `weight` but also caused an undesirable increase in `stress` (as indicated by the high likelihood values). This latter inference bolsters the hypothesis in the second cluster that the designer is attempting to reduce `stress` through material changes in Events 318 and 321 (as reflected by the `PREVIOUSLY-COUNTERED` evidence), even though doing so may counteract the `weight` decrease of the previous cluster.

8.3 Tradeoffs in QIC Usage

Domain knowledge encoded as QIC tables enables the system to generate richer and more meaningful rationale than does the log of recorded design events in isolation. However, the production

[5] A change in catalog number corresponds to selection of a different part from a part catalog.

of the QIC tables by a domain expert can be time-consuming; the background knowledge that a human designer brings to bear when evaluating a design is so rich that to encode more than a small part of it can be impractical. In our system, for example, the QIC tables encode only the impact of changes to attributes or parameters of parts, although in principle the approach could be readily extended to the encoding of structural changes. In addition, the resulting tables are not necessarily reusable across domains because they embed assumptions about the context in which operations will take place and the magnitude of changes.

By using a combination of contextual evidence and background knowledge, rationale extraction can proceed along the range from knowledge-free to knowledge-intensive, depending on what is most practical for a given situation. However, in order for the knowledge-free extraction to be effective, the system must at least have access to the designer's analysis-related activity.

9 Accessing the Rationale: Designer Interaction

Effective use of constructed rationale requires presentation of the information to a designer in a concise and understandable manner. The information can be presented from two perspectives:

Summary Mode Information is presented as a general overview of the design process: What did the designer do, and when (process history)? What different decisions did he or she make, and why? What operations have been performed on this part? What operations affected this design requirement?

Goal-directed Mode Information is presented to address specific issues, related to the goals of a designer who wishes to update or redesign an existing artifact: Are there any dependencies between this part and that one? What parts/subassemblies are most important for this design requirement? What tradeoffs were made between this requirement and that one? What effect do changes on a part have on the design? How important was a given requirement in the overall design (priority)?

The rationale constructed by RCF would be valuable during the development of the initial design, providing the means by which a designer can keep track of options that have been explored and reasons for certain modifications. In this case, access would be focused on answering questions that target specific issues. As such, we would expected goal-directed access to dominate.

We anticipate the extracted rationale to be of even greater use to designers downstream in the life cycle of the design. Effective redesign or maintenance will require that the downstream designers have a deep understanding of the tradeoffs and issues explored by the original designer, as well as key implicit assumptions embodied in the design. Designers who were not involved with the original design would benefit the most; however, the original designers may also need to refresh their memories of the original design process. For redesign and maintenance, goal-directed access would certainly be important. However, the designers would first need to develop a broad understanding of the process involved in the original design, as provided by the summary mode of access.

RCF provides a hierarchical query interface for extracting summary information about a design session. Designers can obtain an overall summary of the design in the form of activity phases grouped by version. They can select a particular version and examine it at the level of activity phases or part-level operations. Refinement phases can also be examined at the level of change clusters (which provide hypotheses of, and evidence for, designer intent), or at the design event level. Refinement phases can be further filtered to examine only change clusters pertinent to a given design requirement, or a given part. At the event level, RCF provides the option to output tabular data, suitable for graphical presentation, showing the history of changes to analysis and parameter values. Changes between different versions of the design can be shown either as text or in graphical form, highlighting pertinent parts in the CAD file itself. A color-coded effort distribution for the different parts in the design can also be mapped onto the CAD model.

Information about parts and part histories can be accessed. The user can query both the final state of a part and its history: its source (a standard or library part, a modified copy of another part, a part designed from scratch), when (which version) it was introduced into the design or removed from the design, whether it was a replacement for another part, modifications to the part, and the effects of those modifications on design requirements, if any. The system also reports any detected dependencies between the part of interest and other parts. *Versions* of parts (*i.e.*, a part that was copied from another part and then modified) can be traced back, along with changes from version to version. The history of changes to a particular parameter or attribute of a part can be presented either textually or graphically.

10 Evaluation and Discussion

The motivation for building RCF was to determine the extent to which nonintrusive methods could extract useful rationale with minimal disruption to the design process. Although no formal evaluation has yet been undertaken, results from the application of RCF to the robotic arm design case represent a qualified success: useful rationale was extracted and represented in structures that provide ready user accessibility. However, additional mechanisms will be required to produce a deployable tool for designers.

10.1 Future Work

The inclusion of additional design metaphors, with particular focus on identifying intent, constitutes one important direction for extending RCF. For example, an *Exploration* metaphor would track branching into different versions of a design from the same source, thus generalizing the linear metaphor currently used. In addition, one might want to determine the priority of different requirements to the designer. Evidence about priority comes not only from how much effort was spent on a requirement, but when in the session that effort was expended. Was effort spent mostly in the beginning, or mostly at the end, or was it distributed throughout the session? The last case may indicate that this requirement was highly significant, and that the designer spent much of his or her time balancing other requirements relative to it.

The inspiration for RCF was the observation that many of the operations that a designer can perform with a CAD tool have meaningful semantic content. As CAD tools increase in sophistication, the set of semantically meaningful operations will increase further, thus enabling additional automated rationale extraction. For example, Active Catalog (Ling et al., 1997) provides a rich query interface for selecting parts from online libraries, as well as simulation capabilities to support a “try before you buy” model of interaction. Observation of the queries formulated by a designer interacting with such a tool would yield a rich data stream from which to infer rationale.

10.2 Knowledge Dependence

There is a fundamental tradeoff in the design of knowledge-based systems between the cost of adding more knowledge (in terms of knowledge acquisition and maintenance) and the value that the added knowledge brings to the problem-solving process. We intentionally designed RCF with an *incremental* knowledge model that enables the system to run with varying levels of domain-specific knowledge. The main sources of application-specific knowledge within RCF are explicit design requirements and the QIC tables. The system can operate without this information, but generates increasingly better results as more of it is provided.

Within RCF to date, design metaphors are domain-independent. However, extraction of more intent-oriented rationale will most likely require the addition of domain-specific metaphors that can directly link actions to explanations.

For domains involving many one-off designs, development of extensive background theories will not be justified. However, for domains in which designs will be repeatedly produced, the application of domain-specific knowledge could greatly increase the extent of the rationales that can be generated.

10.3 Intrusiveness and Usage Requirements

The stated objective for this work was to generate useful design rationale without substantially interfering with or modifying the normal operations of a designer. RCF strongly satisfies the requirement for nonintrusiveness: it runs completely in the background without any interruption to the design process. One could argue, however, that to produce useful rationale information, some limited changes in the design process are required.

Within the current system, the designer must adhere to a small number of conventions in order to support the use of RCF. For example, when creating a new version of an existing design file, the designer must first make a copy of the file and then begin changes on the copy. Changes cannot be made directly on the original design file and then saved to a new file (*i.e.*, via a *Save As* capability), since RCF would incrementally modify its internal model of the original artifact in step with the designer's changes, and then be left with no way to undo those changes when the user saves the artifact as a new version. Conventions of this type amount to minor restrictions on the kinds and order of operations that the designer can perform, and could be eliminated through slight changes to RCF or the CAD system itself. In this case, for example, eliminating the *Save As* capability from the CAD framework would prevent violations of the convention. Similarly, RCF could be readily extended to rollback changes when such Save As commands occur, and recreate them on a copy of the original file.

A more contentious issue relates to RCF's reliance on semantic annotations for objects, as assigned through our extensions to MicroStation95. While RCF can operate without them, the semantic annotations are essential for enabling certain aspects of the rationale extraction. In particular, the *Substitution* metaphor and the effects-based reasoning about designer intent (Section 8.2) require knowledge of the semantic types of objects. As noted in Section 2.1, this type of semantic information is generated as a by-product of parametric design methods and part selection capabilities found within numerous state-of-the-art systems. Thus, while designers are required to perform extra steps to supply annotations within our framework, we feel that such steps will be eliminated in future CAD systems. Of course, the rationale extraction methods that require semantic annotations could not be used in portions of designs that required novel types of components, which will not be predefined in libraries or as parametric models.

11 Related Work

11.1 Models of the Design Process

RCF bases its rationale extraction primarily on general metaphors for designer activity, rather than on domain-specific knowledge about the artifact being designed. These metaphors are intended to reflect the process that designers employ when working. Here, we describe efforts by others to model design processes.

The *task/episode accumulation* (TEA) model of the mechanical design process was derived from protocol analysis of videotaped design sessions (Ullman et al., 1988). This model focuses on *nonroutine* design, whereby the designer has the requisite knowledge and experience to produce the design but the specific task is new. The work was not oriented exclusively toward the automation of the process, but more generally toward understanding the design process and its implications for education and the development of designer aids and work environments. The results suggest that designers satisfy, rather than optimize, and that they do not work from a global plan, but rather develop a central concept from which to start, and then refine the concept locally, on the task and episode level.

The Generic Task Model of Design (GTMD) models the design process independently of the task domain (Brazier et al., 1997). GTMD applies at all levels of design, from conceptual to detailed. In this model, design consists of three types of manipulation subtasks: of requirements, of the design process, and of the design object. Each of these subtasks can be decomposed into *modification*, *update* of modification history, *deductive refinement* (deriving properties, behaviors, and so forth from the current description), and *update of current description*.

The TEA model and GTMD both address the full span of individual designer activity, from receipt of the initial specifications (possibly including the requirements of other parties on the design team) to the production of the final design. In contrast, RCF considers the designer only through the medium of the CAD tool. Relative to the TEA model, RCF has access to the layout and detail tasks, with some access to the catalog selection task. Information about the conceptual design is deemphasized. The TEA model and the GTMD have distinctions between the acts of generating options, analyzing and evaluating these options, and deciding among them. Again, RCF has access to the analysis (and to a certain extent, the evaluation); decisions that take the form of patches, redos, or replacements can also be accessed. However, the generation step is less accessible. More sophisticated access to the catalog selection task, such as the designer using an online catalog that supports interactive queries (*e.g.*, (Ling et al., 1997)), would provide deeper insight into the selection task.

11.2 Designer Queries

(Kuffner and Ullman, 1990) presents the results of an experiment in which designers were given a design and its documentation as well as access to an expert who was familiar with the design. The designers were then recorded while performing a redesign task. The purpose of the experiment was to determine the kinds of questions that designers asked, and how they found the answers. The authors noted several interesting results. Questions about the design tended to be on the component or feature level two-thirds of the time, with the remaining one-third of the time divided more or less evenly between questions about the assembly as a whole, interfaces, or relationships between various components. The designers asked questions about both the original design (*i.e.*, information that is more or less available in traditional documentation), and how potential changes could affect the design. Questions often focused on structure and location more than the operation or purpose of an object; however, when the designer did have operation or purpose questions, it was the human expert who most often was the source of information. In fact, although almost half the questions/conjectures posed by the designers went unconfirmed, or were confirmed by the designers' internal knowledge, the great majority of the remaining questions were answered by the expert. As the authors say:

> The fact that ... designers referred to the examiner's stored knowledge at all indicates that mechanical designers would use design information stored in any intelligent CAD tool, if available.

These results suggest that "intelligent CAD tools" should concentrate on providing operation and purpose information, where possible, because traditional documentation omits them. They also suggest that designers would use such a tool even for information already stored elsewhere, if the information could be readily extracted (as from a human expert).

Based on a similar survey of designer protocol data, (Gruber and Russell, 1996) concludes that design rationale tools should concentrate on collecting a broad range of pertinent data about a design. This data should be retrievable in response to queries in such a way that rationale can be reconstructed from the data. They call this the *generative approach* to design rationale.

11.3 Design Rationale Acquisition and Retrieval

Early work on the acquisition of design rationale focused on *direct solicitation* methods, whereby users are explicitly queried or provided with structured interfaces to elicit rationale (Russell et al., 1990; Gruber, 1991). Because these approaches impose a heavy documentation burden on designers, they have had limited success.

Several attempts have been made to ease the problem of automated rationale generation by imposing structure on the design process (Ganeshan et al., 1994; Brazier et al., 1997). The general idea is to model design as selection from predefined transformation rules. Rule selections are recorded along with rule-specific rationale. By structuring the design process, these approaches can provide deeper explanations of designer intent than can RCF. However, they greatly constrain designer activities and are unsuitable for *ad hoc* design cases requiring novel design methods.

For example, (Ganeshan et al., 1994) describes a system in which the design is generally manipulated through predefined transformations on objectives. To manipulate the design directly, a designer must provide explicit justifications to the system. The design history consists of the log of transformations, while designer intent is is described by the mapping between transformations and objectives. Similarly, the work in (Brazier et al., 1997) models design as selection from predefined transformation rules that include rationale information related to the operations performed by the rule. Upon selection of a rule, the choice is recorded along with the rule-specific rationale. The paper provides an example of how to use the produced rationale to support redesign, related to the design of the Fokker 60 (Fo60) passenger aircraft from the design of the Fo50.

The Active Design Document (ADD) system (Garcia and Howard, 1992; Garcia et al., 1997; Garcia and de Souza, 1997) generates rationale for *parametric design tasks*, in which the design process involves constraining or selecting values for fixed sets of parameters. This class of designs, while important, is much narrower in scope than that addressed by RCF. The early ADD system observed the designer's actions, and then attempted to generate explanations for them by using a predefined knowledge base for the domain. If the system was unsuccessful, or predicted an action other than that taken, it would ask for explanation. In this way, a knowledge base could be built from observed cases. More recent versions of ADD can generate ranked alternatives and allow the designer to change the knowledge base to update the system's reasoning abilities.

(Qureshi et al., 1997) describes a system for the nonintrusive archiving, and subsequent querying, of electromechanical design process histories. Its *Integration Core Model* representation of design history uses both a generic process model and specialized process models to represent the design information. The generic process model is independent of the task domain (within the larger domain of electromechanical design) and the type of design task performed (*e.g.*, conceptual, parametric). Generic process history is essentially a step-by-step record of all activity. Local specialized models, which use more specific domain knowledge to structure low-level detailed information about various aspects of the design, can be incorporated into the system. In particular, design rationale (in the sense of designer intent) can be formulated as a specialized model, separate from the Core Model. This approach allows different domain-specific models of rationale to be incorporated into the archiving process.

The Design History Tool (Chen et al., 1991) evolved from TEA model of the design process and the studies of Kuffner and Ullman on the understanding of others' designs. This system stores structured, hierarchical representations of a design, which are extracted from manually transcribed videotapes of design sessions that include what the designer *says* as well as the operations he or she performs. The resultant design history can be browsed with respect to structure, evolution, alternatives considered, and dependencies in the design or among the requirements. This system provides more direct access to the designer's thought processes and intentions than does RCF, but at the costs of intrusiveness into the design process and labor-intensive data input.

12 Conclusions

RCF's methods for acquiring design rationale present an innovative approach to a difficult and important problem. Previous work on rationale acquisition has focused on highly intrusive techniques that either involve extensive participation by the human designer or change the underlying design process. Because of the level of disruption to the underlying design process, tools of this type have not been embraced within the design community.

In contrast, our idea of extracting design rationale from observations of designer activity is rooted in a philosophy of nonintrusiveness: rationale is produced as a natural by-product of the design process. The human designer will need to intervene to supply certain information of relevance to the design but should be relieved of responsibility for recording information about the noncreative aspects of a design. We have shown that this type of automation is possible by applying key AI methods such as structured knowledge representation, knowledge-based plan recognition, clustering techniques, and basic qualitative reasoning.

Acknowledgments

This research has been supported by DARPA Contract N00014-96-C-0162. Kurt Konolige and John Peters contributed to the development of the RCF system.

References

Bentley Systems, Inc. (1995). *MicroStation Modeler User's Guide*. Version 5.5.

Brazier, F. M., van Langen, P. H., and Treur, J. (1997). A compositional approach to modelling design rationale. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 11:125–139.

Carroll, J. M. and Moran, T. P. (1991). Special issue on design rationale. *Human-Computer Interaction*, 6(3-4).

Chen, A., Dietterich, T. G., and Ullman, D. G. (1991). A computer-based design history tool. In *NSF Design and Manufacturing Conference*, pages 985–994, Austin, Texas.

Conklin, E. J. and Yakemovic, K. B. (1991). A process-oriented approach to design rationale. *Human-Computer Interaction*, 6:357–391.

De Kleer, J. and Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, 24:1–83.

Fischer, G. and Lemke, A. C. (1988). Construction kits and design environments: Steps toward human problem-domain communication. *Human-Computer Interaction*, 3:179–192.

Fischer, G., Lemke, A. C., McCall, R., and Morch, A. I. (1991). Making argumentation serve design. *Human-Computer Interaction*, 6:393–419.

Ganeshan, R., Garrett, J., and Finger, S. (1994). A framework for representing design intent. *Design Studies*, 15(1):59–84.

Garcia, A. C. B., de Andrade, J. C., Rodrigues, R. F., and Moura, R. (1997). ADDVAC: Applying active design documents for the capture, retrieval, and use of rationale during offshore platform VAC design. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 986–991.

Garcia, A. C. B. and de Souza, C. S. (1997). ADD+: Including rhetorical structures in active documents. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 11:109–124.

Garcia, A. C. B. and Howard, H. C. (1992). Acquiring design knowledge through design decision justification. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 6(1):59–71.

Gero, J. S. (1990). Design prototypes: A knowledge representation schema for design. *AI Magazine*, 11(4):26–36.

Gruber, T. R. (1991). Interactive acquisition for justifications: Learning 'why' by being told 'what'. *IEEE Expert*, 6(4).

Gruber, T. R. and Russell, D. M. (1992). Generative design rationale: Beyond the record and replay paradigm. Technical Report KSL 92-59, Knowledge Systems Laboratory, Computer Science Department, Stanford University.

Gruber, T. R. and Russell, D. M. (1996). Generative design rationale: Beyond the record and replay paradigm. In *Design Rationale: Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Mahwah, New Jersey.

Hutchins, E. L., Hollan, J. D., and Norman, D. A. (1986). Direct manipulation interfaces. In *User Centered System Design, New Perspectives on Human-Computer Interaction*, pages 87–124. Lawrence Erlbaum Associates Inc., Hillsdale, NJ.

Iwasaki, Y. and Simon, H. A. (1986). Causality in device behavior. *Artificial Intelligence*, 29(1):3–32.

Iwasaki, Y. and Simon, H. A. (1994). Causality and model abstraction. *Artificial Intelligence*, 67(1):143–194.

Kuffner, T. A. and Ullman, D. G. (1990). The information requests of mechanical design engineers. *Design Studies*, 12(1):42–50.

Ling, S. R., Kim, J., Will, P., and Luo, P. (1997). Active catalog: Searching and using catalog information in internet-based design. In *Proceedings of the ASME Design Engineering Technical Conferences*.

Moran, T. P. and Carroll, J. M. (1996). Overview of design rationale. In *Design Rationale: Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Mahwah, New Jersey.

Qureshi, S. M., Shah, J. J., Urban, S. D., Harter, E., Parazzoli, C., and Bluhm, T. (1997). Integration model to support archival of design history in databases. In *Proceedings of the ASME Design Theory and Methodology Conference*.

Rittel, H. and Webber, M. (1973). Dilemmas in a general theory of planning. *Policy Science*, 4:155–169.

Russell, D. M., Burton, R. R., Jordan, D. S., Jensen, A. M., Rogers, R. A., and Choen, J. (1990). Creating instruction with IDE: Tools for instructional designers. *Intelligent Tutoring*, 1(1):3–16.

Shipman, F. M. and McCall, R. J. (1997). Integrating different perspectives on design rationale: Supporting the emergence of design rationale from design communication. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 11:141–154.

Ullman, D. G., Dietterich, T. G., and Stauffer, L. A. (1988). A model of the mechanical design process based on empirical data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2(1):33–52.