AutoCRAT: An Automated Tool to Support Conflict Detection in a Concurrent Engineering Environment

Donald R. Schwartz
Lois M. L. Delcambre

Technical Report No. CS/E 94-021

May 1994

AutoCRAT: An Automated Tool to Support Conflict Detection in a Concurrent Engineering Environment

Donald R. Schwartz
Department of Computer Science
Louisiana Tech University
P. O. Box 10348
Ruston, LA 71272
(318) 257-2621
schwartz@engr.latech.edu

Lois M. L. Delcambre
Dept. of Comp. Science and Engineering
Oregon Graduate Institute of Science & Technology
P. O. Box 91000
Portland, OR 97291-1000
(503) 690-1689
lmd@cse.ogi.edu

Abstract

Concurrent engineering requires the simultaneous consideration of a huge diversity of issues regarding a given object under design. Successful support for concurrent engineering requires a conceptually integrated environment that can address this diversity of issues and assist with the detection and resolution of conflicts among design constraints and decisions.

The Automated Constraint Refinement and Assessment Tool (AutoCRAT) is a generic tool for the top-down refinement and bottom-up validation of design constraints. AutoCRAT supports the diversity of issues that arise in a concurrent engineering environment because constraints can address any aspect of the product under design and because constraints can be expressed in any language, including natural language.

The focus of this paper is on the structuring facility of AutoCRAT that supports a user-defined organizational structure for constraints. This structure allows constraints to be flagged, annotated, browsed, and queried. This paper describes how the structuring facility of AutoCRAT supports conflict detection.

1. Introduction

Concurrent engineering refers to the collaborative effort of teams of specialists who each focus on a distinct aspect of the object (or system or product) under design [Shi91].

There are several challenges that must be addressed to support concurrent engineering.

(1) The design environment must be able to simultaneously address all of the diverse aspects considered for concurrent engineering [RWC91]. Examples of the various aspects include: the cost of, the ability to manufacture, and the ability to repair or maintain the object, in addition to the usual aspects concerning form (e.g., weight, size) and function (e.g., gas mileage, power output).

(2) The design environment must support the entire life cycle of the object [Tur91]. Each aspect considered during concurrent engineering has, historically, been considered at a particular stage in the life cycle. Aesthetics might be part of the earliest design (e.g., when design sketches are done). Form might be considered fairly early in order to set the physical dimensions of the object. Function soon follows (or may proceed in parallel). Manufacturability may (or historically may not) have been considered very early in the process. Safety and cost might be considered throughout the life cycle.

A design environment to support concurrent engineering must be able to support all aspects through the entire life cycle so that conflicts from all sources can be

identified, examined, and resolved whenever they occur.

(3) The design environment must interoperate with a large and growing number of design tools that can provide sophisticated assistance for some particular aspect of the design. Such specialized tools might provide solid models, stress analysis, safety, simulation testing, mechanical drawing, etc.

The goal, then, is to provide a conceptually unified environment that supports the diversity of aspects and design tools considered and used during concurrent engineering throughout the entire life cycle of a product. The question is: what is an appropriate generic framework for such a design environment?

This research adopts the Theory of Plausible Design (TPD) [AD87] as the generic framework for a design environment. TPD represents the design, at any stage of the life cycle, as a set of constraints, and supports a design process of top-down refinement of those constraints. Perhaps the most valuable aspect of TPD is that it also provides for the validation (or refutation) of design constraints through the provision of evidence. Evidence for a detailed (bottom-level) constraint establishes the plausibility state for the constraint. The plausibility state (of either validated or refuted) is then propagated upwards through the refinement graph to ultimately validate (or refute) a complete design. TPD supports concurrent engineering because constraints stated in TPD can be expressed in any language (including natural languages) and can address any aspect of the object under design. TPD can easily represent "The car must be cute and sporty" as well as "The car must deliver 140 hp."

This paper focuses on AutoCRAT [DS92], an automated tool that implements TPD. AutoCRAT assists human designers through the articulation of constraints, the top-down

refinement of constraints into alternative sets of subconstraints, and ultimately through the documentation of evidence that the designer uses to establish the likelihood that the constraint will be satisfied when the object is actually constructed (i.e., to validate or refute the constraint).

The diversity of issues addressed during concurrent engineering significantly increases the likelihood of conflict and increases the difficulty of conflict detection. The goal of concurrent engineering and the critical element of its success is the detection and resolution of conflict. Also, the benefit is greatest when such conflicts are detected early. A rule of thumb during the design of space systems is that 90% of the budget is committed before 10% of the budget is actually spent. This implies that early detection and resolution of conflict has a huge potential benefit for cost control.

AutoCRAT represents constraints, the refinement structure, evidence, and the plausibility states in order to support TPD. AutoCRAT provides a structural facility for constraints that can be used to flag or mark and annotate constraints (e.g., that address a given aspect of the design) and formulate queries based on Boolean combinations of flags. This paper explains how the AutoCRAT structuring facility helps the design team detect conflict. Note that AutoCRAT allows the user to define and refine constraints, and provide evidence for constraints; AutoCRAT does not currently do any automatic design. Similarly, the AutoCRAT structuring facility allows design teams to detect and resolve conflict.

The organization of this paper is as follows. Section 2 provides the background for AutoCRAT with a brief presentation of TPD. Section 3 presents the AutoCRAT structuring

facility in detail, along with examples. Section 4 describes how the structuring facility supports conflict detection among constraints. Conclusions and future work are offered in Section 5.

2. Background: The Theory of Plausible Design as the Basis for AutoCRAT

The Theory of Plausible Design (TPD) is intended to capture the essence of design, particularly in the earlier stages of the life cycle. This section introduces TPD in Section 2.1 and explains the AutoCRAT support for TPD in Section 2.2.

2.1. Overview of the Theory of Plausible Design (TPD)

Early in the design, requirements are usually stated in natural language, and as such, may be expressed in imprecise, ambiguous, and/or non-quantitative terms such as "efficient", "manufacturable", and so on. Yet, the design team, at each stage of the design, must begin with such specifications and produce a more-detailed, more-precise design, i.e., a description, specification or plan that represents the target artifact. At the final stage, the design serves as a blueprint for manufacture, and an artifact manufactured according to the design is expected to satisfy the given set of requirements. The challenge is to produce a precise, final design that the designer has confidence will satisfy the imprecise requirements given as a starting point.

The central notion of the Theory of Plausible Design (TPD) is that the process of design is a process of top-down refinement of a set of *constraints*, properties that must be satisfied by the artifact under design. Constraints can address the functionality, performance,

reliability, safety, cost, aesthetics, manufacturability, modifiability, etc., of the product under design. In practice, when constraints of several types must be considered together, e.g. in a concurrent engineering environment, conflicts among the constraints often arise. For example, design constraints concerning safety and reliability generally cause problems for constraints on cost and manufacturability. When additional constraints concerning weight and power requirements are introduced, additional conflicts can arise.

The process of design is often a process of refinement. For each design statement the designer must answer the question: "How can I demonstrate that this constraint is satisfied?" Often, the answer results in a new, more detailed set of design subconstraints. TPD captures this process of refinement by building a constraint dependency graph (CDG) in a top-down manner to document how the design process proceeded as the constraints were formulated. The CDG can include design alternatives through the use of **or** and refinement into a set of subconstraints through the use of **and**. In addition to providing a set of subconstraints, a given constraint can be validated or refuted by the provision of evidence that directly demonstrates that the constraint is, or cannot be, satisfied.

One of the driving factors that led to the development of TPD is that a designer, at any stage in the design process, would like to have confidence that the original, top-level requirements of a design are being met. The goal is to validate a complete design by validating the original set of top-level constraints. The initial state of constraints, as they are entered into the CDG, is *assumed*. The validation process begins by validating the leaf-level or most-detailed constraints directly through the provision of evidence. The

evidence provided for a leaf-level constraint documents the reasons which either support or defeat the constraint. Based on the evidence, the state of the constraint is either *validated* or *refuted*, respectively. Stated more formally, a constraint C can be in exactly one of three plausibility states:

Assumed: If no evidence against C's plausibility has been documented.

Validated: If significant evidence in favor of and no evidence against C's plausibility has been documented.

Refuted: If evidence against C's plausibility has been documented.

The state of each leaf-level constraint is then systematically propagated upwards based on the and/or connections in the graph according to the laws of plausibility [Das89].

The plausibility state of a constraint is thus a function of the available evidence. TPD provides for the documentation of evidence, which can be of any type. Examples of evidence include: formal proofs, which uses formal deductive logic to demonstrate that the plausibility state of a constraint is implied from given premises; experiments, from which conclusions can be drawn; data, analyses or models reported in the technical literature; heuristics, arguments based on common sense reasoning; expert opinion, given by by one or more experts in the realm of the particular design; previous experience; simulation; testing; and prototyping.

The decision about what constitutes the end of a design varies depending on the point of view of the designer. Early design may begin with an initial set of requirements and produce a more detailed set of requirements as the final, leaf-level deliverable. This in turn becomes the starting point for the next team of designers. Each design phase can use

TPD to support the top-down development and bottom-up validation of the design.

2.2. Overview of AutoCRAT

TPD was adopted as the formal framework for a design tool called AutoCRAT: the Automated Constraint Refinement and Assessment Tool. AutoCRAT has been implemented with inspiration provided by the early design stages for satellite and space systems. [DSG92] Space systems tend to be one-of-a-kind or at least few-of-a-kind, yet they require sophisticated functionality under the pressure of limited budgets and precise schedules.

AutoCRAT provides four windows, including both a textual and a graphical representation of constraints as they are organized into the CDG: the Constraint Graph window shows the top-down refinement structure of constraints and the Notebook window displays a series of menus and forms to provide for the entry, display and manipulation of all textual data, including the statement of constraints and evidence posted against (leaf-level) constraints. A third window, the Message window, is used to display error messages and other information originating in any of the components of AutoCRAT. The fourth window is called the Flag Graph window. The capabilities of flags and their associated annotations to organize and retrieve constraints is the major focus of this paper. These capabilities are referred to as the AutoCRAT structuring facility and are explained in the following section.

3. The AutoCRAT Structuring Facility

In addition to the definition and maintenance of constraints within the constraint dependency graph, a structural facility has been provided in AutoCRAT that allows the designer to organize, retrieve, and further describe constraints that appear in the constraint dependency graph [Sch93]. The structuring facility provides a way to organize and annotate constraints. The query facility uses this structure as a basis for browsing and highlighting constraints.

3.1. The AutoCRAT Flag Facility

There are two uses of flags in the structural facility: (1) to identify constraints which fall into particular, user-defined classes and (2) to enter additional textual information that the designer wishes to capture about individual constraints.

The use of flags to mark constraints is introduced here with an example. There may be a certain set of constraints that are particularly important to the overall cost of an object under design. A flag can be defined with the name "Cost". Each constraint that affects cost can then be marked with this flag. There may be another flag defined with the name "Safety" to tag constraints that pertain to some safety aspect of the object under design. Any category of interest can be defined as a flag and can then be used to mark constraints.

The flags can be organized into a simple semantic network, meaning that subflags and sub-subflags, etc., can be defined. A subflag is a flag that represents a more specific concept than its parent flag. As an example, when designing a car, a flag might be defined

called "Driver Safety", meaning that all constraints that are considered to be critical to the safety of the driver are marked with the flag "Driver Safety". Another flag could be defined to mark those constraints concerning passenger safety. These flags, "Driver Safety" and "Passenger Safety", could be defined as subflags under the "Safety" flag. Graphically, a subflag appears to the right of its parent flag.

Based on the semantics of subflag, whenever a constraint is marked by a flag, it is automatically marked by all of its superflags as well. Under the configuration of flags in Figure 1, any constraint marked by "Driver Safety" would automatically be marked by both the "Driver Safety" and "Safety" flags.

The other use of flags allows that each time the flag is associated with a constraint, the designer has the opportunity to enter a textual description about the association. One example of this use of flags is the way that AutoCRAT supports the documentation of evidence. There is a flag for "Evidence" and there are subflags for each type of evidence needed during the design. Examples of evidence include, but are certainly not limited to, previous experience, simulation, expert opinion, and testing, as shown in Figure 2.

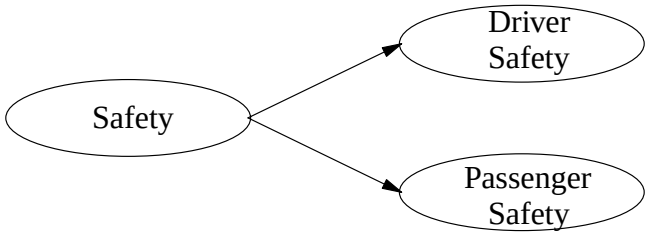


Figure 1. Safety Flags

For each subflag of evidence, the designer can enter a description giving the details of the evidence provided. As an example, if constraint C70 were validated because an existing, available component already satisfied that requirement, and this use of the component requires no changes, then C70 could be associated with the "Previous Experience" flag. The annotation or documentation for this evidence entry might be "The design used in a previous model exhibited excellent performance and required only routine, scheduled maintenance." In space systems, such evidence is used for validation when a component has successfully "flown before", e.g., on a previous shuttle mission. If another constraint, say C81, were validated using simulation results, then C81 could be associated with the "Simulation" flag and the textual description might say: "Simulation program model-1243 for the given car provides evidence that this constraint is satisfied with a confidence level of 0.95."

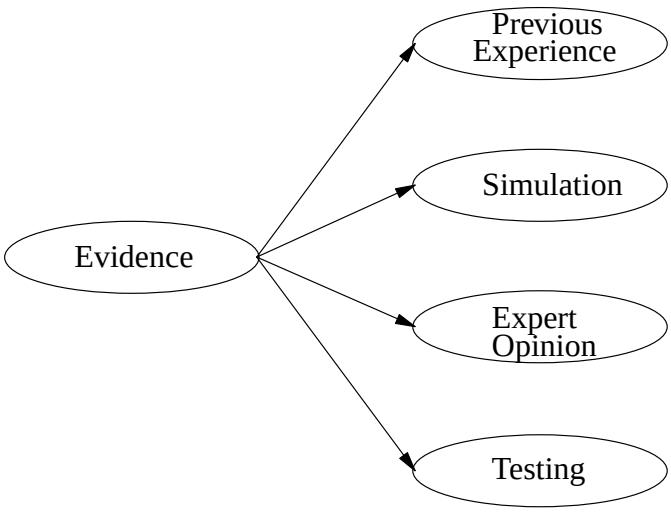


Figure 2. Evidence Subflags

The use of annotations with flags can also provide additional structure or attributes for the constraint itself. It might be useful to describe several different aspects of the constraint. Such annotation can be organized by providing an appropriate flagging structure that allows constraint-specific information to be entered according to the various categories established by the flagging structure. This use of flags might be called a "table of contents" for a constraint. As an example, consider the flagging structure in Figure 3.

For each of the flags that appears below the flag called "Constraint", the user/designer can enter textual information specific to the constraint. The designer can thus enter the justification for including the constraint, independently from the actual constraint statement. Also, the designer can enter the expected feasibility of the constraint, the source of the constraint, the author of the constraint (with any commentary), and the justification for including the constraint in the design. Each AutoCRAT user, as well as each application area, can custom design the flagging structure to serve as a "table of contents" for constraints to meet his/her specific needs. Using other terminology, each flag (with annotation) serves as an attribute for constraints. Thus a flag with subflags serves as a hierarchically-structured attribute for constraints.

The decision about when, if ever, to enter the constraint-specific information (i.e., the annotation) for a given association of a flag with a constraint is left to the user. Some

Figure 3. Sample Table-of-Contents Flag Structure

constraints may be associated with the flag "Simulation" long before the details are known. Thus, it is up to the user to enter the constraint-specific annotation whenever desirable.

3.2. The AutoCRAT Querying Facility

The flagging facility provides the framework for querying the design. The AutoCRAT querying facility allows the designer to browse and query against the constraint dependency graph, including the annotations [SD93]. AutoCRAT provides pre-defined queries for several commonly-used queries and a general query facility based on flags.

The AutoCRAT predefined queries include such queries as "List all constraints and their states" and "List all flags associated with a particular constraint". The flag-based query facility allows the user to select flags and combine them using using the connectors AND, OR, AND NOT, and OR NOT to develop ad hoc queries. These queries are built using mouse clicks on alternating lists of flags and connectors. For example, if the user wanted to see all constraints validated by H. Brown, he would build the query using

validated AND H. Brown

He/she may wish to see all constraints which deal with cost and either power or performance:

cost AND power OR cost AND performance

To see constraints that affect cost, but that are not related to safety, the user can issue the following query:

cost AND NOT safety

In addition, queries can use a string searching capability based on the actual text of annotations, including the constraint statements themselves. This allows the AutoCRAT user, for example, to search for all constraints that mention the terms mileage, gas, economy, or MPG in the constraint statement. Such selection criteria can be combined with other flags or selection criteria using the AND, OR, and NOT connectors.

In response to a query, AutoCRAT returns a list and description of the constraints which satisfy the query. To better orient the user as to which constraints are returned, these constraints are highlighted in the graphical representation of the CDG. Facilities are also present which allow the user to browse through the complete constraint statements for those constraints that comprise the answer to the query.

4. AutoCRAT Support for Conflict Detection

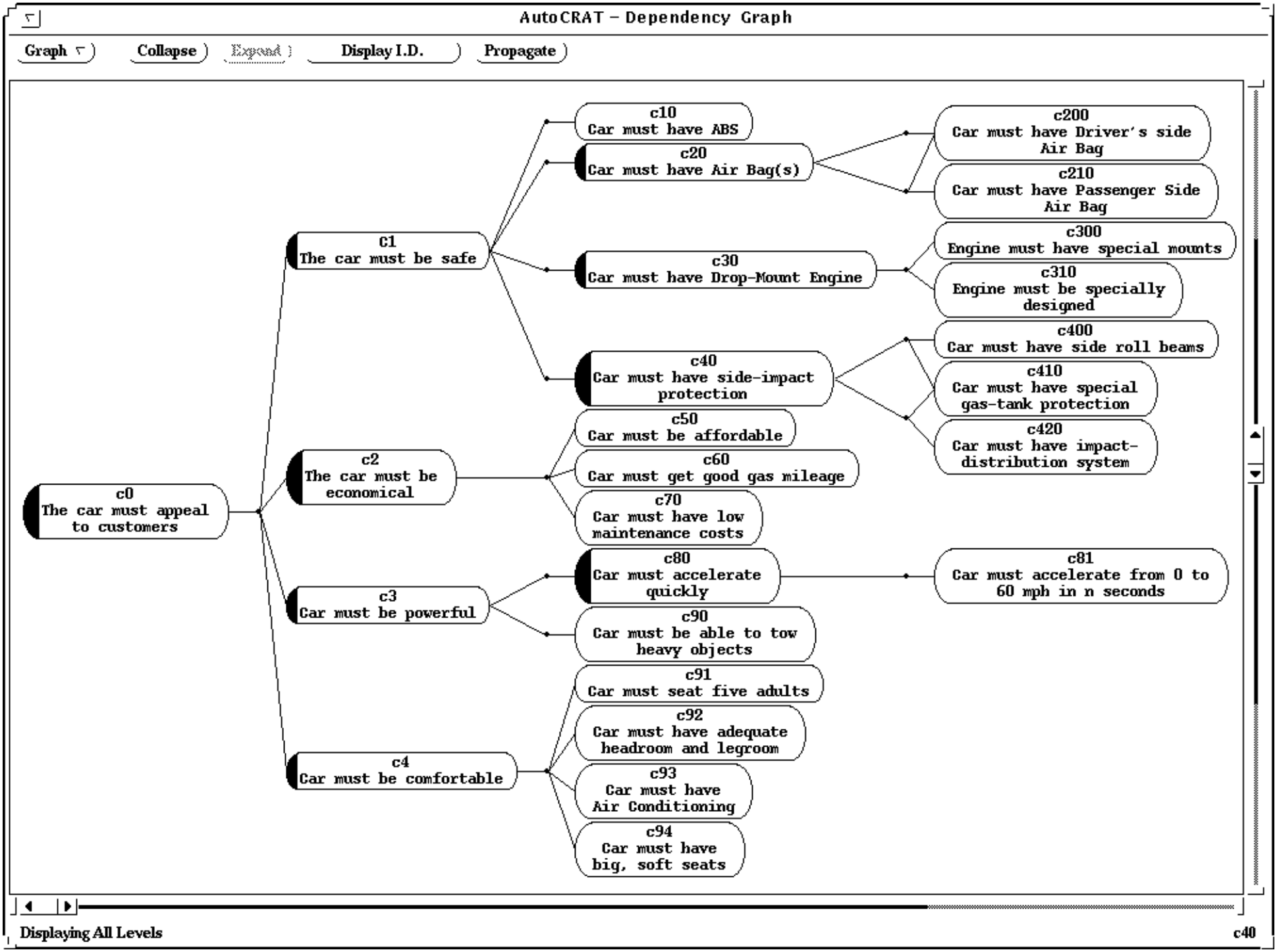
The AutoCRAT support for conflict detection can be viewed as a three-part process: (1) Identification of constraints that (unexpectedly or unknowingly) deal with other, diverse aspects of the design (through the string-matching facility), (2) Identification of all constraints that deal with multiple, particular, overlapping, and potentially conflicting aspects of a design (through the query facility), and (3) Inspection of the design rationale that lead to each (conflicting) constraint (through examination of the refinement process documented in the CDG) and the evidence given to validate and/or refute this and other alternative design choices.

Note that AutoCRAT is a support tool for a human designer. AutoCRAT, as currently defined and implemented, allows the designer to state constraints, refine them, and establish their plausibility state. For conflict detection, AutoCRAT likewise provides the human designer with technology to organize constraints, search for conflicts, and understand the design rationale for each (conflicting) constraint. Thus AutoCRAT does not explicitly provide for automatic design nor automatic conflict detection/resolution. However, AutoCRAT, through the CDG and the structuring facility, can be viewed as providing a design representation structure which could directly support intelligent processing, e.g., for automatic design or conflict detection.

In this section, a rather simple and contrived example of the design of a car is used to demonstrate AutoCRAT's support for conflict detection and resolution. When designing a car, some typical top-level constraints might be:

- The car must be safe.
- The car must be economical.
- The car must be powerful.
- The car must be comfortable.

Naturally, each of these would be refined. A resulting constraint dependency graph (CDG) might appear as follows.


AutoCRAT – Dependency Graph
Graph
Collapse
Expand
Display I.D.
Propagate
c0
The car must appeal to customers
c1
The car must be safe
c2
The car must be economical
c3
Car must be powerful
c4
Car must be comfortable
c10
Car must have ABS
c20
Car must have Air Bag(s)
c30
Car must have Drop-Mount Engine
c40
Car must have side-impact protection
c50
Car must be affordable
c60
Car must get good gas mileage
c70
Car must have low maintenance costs
c80
Car must accelerate quickly
c90
Car must be able to tow heavy objects
c91
Car must seat five adults
c92
Car must have adequate headroom and legroom
c93
Car must have Air Conditioning
c94
Car must have big, soft seats
c200
Car must have Driver's side Air Bag
c210
Car must have Passenger Side Air Bag
c300
Engine must have special mounts
c310
Engine must be specially designed
c400
Car must have side roll beams
c410
Car must have special gas-tank protection
c420
Car must have impact-distribution system
c81
Car must accelerate from 0 to 60 mph in n seconds
Displaying All Levels
c40

The designers would also be required to associate flags with the various constraints. Adding to the flags introduced in Figure 1, the following flag structure in Figure 4 results.

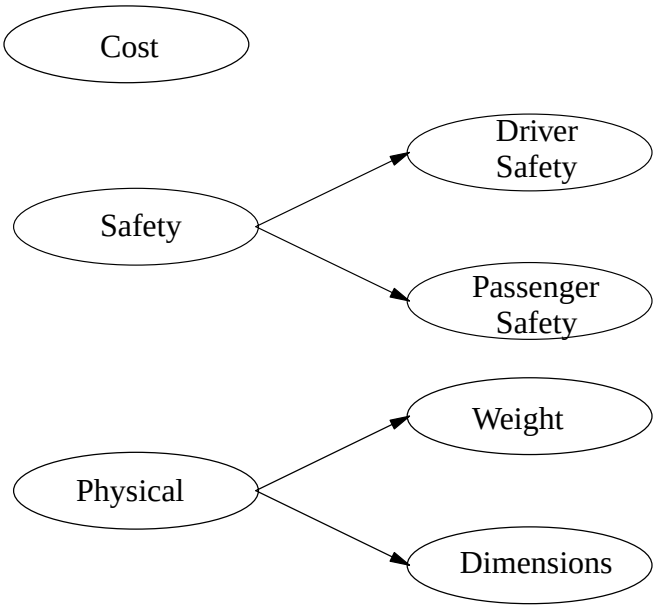


Figure 4. Flags

Sample constraint-flag associations are depicted in the Table 1. Using these constraint-flag associations, a designer (or manager or assessor) can easily create queries to browse the CDGs in order to detect conflicts. For example, to see which constraints affect cost and driver safety, the following query is used:

cost AND driver safety

Table 1: Constraint-Flag Associations

Flag Name	Constraint ID			
Cost	C10	C20	C30	C40
	C50	C60	C70	C93
	C94	C200	C210	C300
	C310	C400	C410	C420
Driver Safety	C10	C20	C30	C40
	C200	C300	C310	C400
	C410	C420		
Passenger Safety	C10	C20	C30	C40
	C210	C300	C310	C400
	C410	C420		
Safety	C10	C20	C30	C40
	C200	C210	C300	C310
	C400	C410	C420	
Weight	C30	C300	C400	
Size	C91	C92	C94	C300
	C400			

The result of this query is the following set of constraint IDs:

C10	C20	C30	C40
C200	C300	C310	C400
C410	C420		

These constraints can then be individually examined to see whether or not they conflict. For example, suppose a constraint exists stating that a particular engine is needed to provide the required power but another constraint limits the total weight of the car to 2000 pounds. Both constraints should be associated with the "weight" flag. If the weight of the proposed engine causes the total weight of the car to exceed 2000 pounds, that engine

would have to be rejected or modified or the total weight constraint would have to be modified (if that were a feasible option). This is precisely the type of conflict that should be detected during concurrent engineering. If the designers look for and evaluate conflicts early in the design life cycle, the project will certainly benefit in the long run.

5. Conclusions and Future Work

Too often, design tools tend to be application specific and most fail to support the additional requirements needed for concurrent engineering. AutoCRAT is a unique tool in the realm of design. It provides a generic facility for the statement, organization, monitoring, and refinement of design constraints throughout the design life cycle. Because no restrictions are placed on the type of constraints, nor the way in which constraints are stated, AutoCRAT allows the designer to consider all aspects of design. Constraints arising from all aspects of concurrent engineering can be entered and monitored just as easily as constraints concerning form and function. Through the use of the flagging facility, the designer can identify all constraints which affect any particular aspect of the artifact. This is a particularly important benefit during the assessment of a design, especially when the design must be approved by multiple assessors: each assessor can trace decisions made about his particular area, e.g., safety or cost.

Because the state of each constraint is automatically propagated upward through the CDG, and the state of each constraint is graphically represented in the CDG, conflicts which arise due to various constraints are clearly indicated (since the state of such

constraints will be "refuted"). Using the various browsing and query facilities available in AutoCRAT, the designer can review the history of the design process, the evidence provided for the various constraints, and the justification given for the inclusion of the conflicting constraints. AutoCRAT helps to **detect** these conflicts, but does not offer advice on how to **resolve** the conflicts.

The contribution of AutoCRAT arises from its focus on: a flexible representation for constraints (based on the flags to define additional fields of interest that can represent statements in any formal or natural language), the generic aspects of design (based on top-down refinement, provision of evidence, and bottom-up validation of the initial requirements), and the organizational and retrieval capabilities (based on the Structural and Query Facilities).

AutoCRAT has been implemented and initially tested in the satellite design arena [SD93]. The long-term research plan for AutoCRAT envisions an architecture where AutoCRAT provides the unified representation of design as a middle layer with various knowledge-based and other tools for automatic design layered on top of AutoCRAT. Below AutoCRAT, one could imagine various specialized design tools such as CAD or modeling tools, that represent specific aspects of the object under design. The key to successful concurrent engineering is in the seamless integration of design tools with the generic constraint statement, validation, and conflict detection capabilities of AutoCRAT. As part of the pilot implementation, AutoCRAT was successfully integrated with VEHICLES, a knowledge-based system for spacecraft design [BG90]. The interface between the two

systems focused on the introduction of several additional predicates (in the VEHICLES Prolog program) that were asserted whenever a certain aspect of the design succeeded. In essence, VEHICLES was posting evidence to AutoCRAT automatically. In general, the interface requires the delivery of constraints to the underlying design tool (in an understandable format) and the automatic posting of evidence back to AutoCRAT.

The major effort of the original research involving AutoCRAT was its integration with a CAD database which was developed in order to provide a generic representation of design object components and configurations and to truly capture the essence of functional block diagrams which describe the objects [BK85][BdC85]. This integration provides the basis for a design environment which supports the top-down refinement of design constraints and the bottom-up configuration of the physical objects being designed [Kat90]. The integration of the generic CAD database with the power of AutoCRAT provides an environment that spans the design life-cycle from high-level, abstract requirements to actual, implemented design object. This environment also allows the user to trace design decisions made and to track the multidimensional set of constraints and requirements which arise in concurrent engineering.

References

[AD87] U. Aguero and S. Dasgupta. A Plausibility-Driven Approach to Computer Architecture Design. *Communications of the ACM*, 30(11):922-932, Nov. 1987.

[BdC85] Alejandro P. Buchmann and Concepcion Perez de Celis. An Architecture and Data Model for CAD Databases. In *Proceedings of the Eleventh International Conference on Very Large Databases*, pages 105-114, 1985.

[BG90] K. L. Bellman and A. Gillam. Achieving Openness and Flexibility in VEHICLES. *Proceedings of the SCS Eastern Multiconference*, Simulation Series,

Volume 22, Number 3, The Society for Computer Simulation, April 1990.

[BK85] D. S. Batory and Won Kim. Modeling Concepts for VLSI CAD Objects. *ACM Transactions on Database Systems*, 10(3):322-346, Sept. 1985.

[Das89] Subrata Dasgupta. The Structure of Design Processes. *Advances in Computers*, 28, 1989.

[DS92] Lois Delcambre and Donald Schwartz. AutoCRAT: Automated Support for Design in a Concurrent Engineering Environment. In *Proceedings of the European Joint Conference on Engineering Systems Design and Analysis*, June 1992.

[DSG92] Lois Delcambre, Donald Schwartz, and April Gillam. AutoCRAT Templates for Design Knowledge Capture. In *Proceedings of the International Space Year Conference on Earth and Space Science Information Systems*, Feb. 1992.

[Kat90] Randy Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4):375-408, Dec. 1990.

[RWC91] Ramana Reddy, Ralph Wood, and K. Cleetus. The Darpa Initiative: Encouraging New Industrial Practices. *IEEE Spectrum*, Vol. 28, No. 7, pages 26-29, July 1991.

[Sch93] Donald Schwartz. A Computer-Aided Design Database Integrated with a Generic, Formal Design Paradigm. Ph.D. Dissertation, University of Southwestern Louisiana, May 1993.

[SD93] Donald Schwartz and Lois Delcambre. Credibility Assessment Using AutoCRAT. In *Proceedings of the 1993 Simulation MultiConference: Simulation in Military and Government*, pages 67-72, Apr. 1993.

[Shi91] Sammy G. Shina. New Rules for Design. *IEEE Spectrum*, Vol. 28, No. 7, pages 23-25, July 1991.

[Tur91] Jon Turino. Concurrent Engineering: Making It Work. *IEEE Spectrum*, Vol. 28, No. 7, pages 30-31, July 1991.