From: AAAI Technical Report SS-00-03. Compilation copyright © 2000, AAAI (www.aaai.org). All rights reserved.

Knowledge representation and management for Engineering Design

Fatma Mili
School of Engineering and Computer Science
Oakland University, Rochester MI48309-4478
mili@oakland.edu

Introduction, Motivation

Engineering design is the process by which product requirements are transformed into product descriptions. Engineering design is distinguished from design in other more purely creative domains by the fact that it is a highly disciplined and highly regulated activity. The highly regulated nature of engineering is at the same time a blessing and a burden. It is a blessing because the discipline and the methodologies safeguard engineers from wasting time and efforts reinventing wheels. A good proportion of engineering work consists of reusing existing solutions and applying predefined algorithms. The high level of regulation is also an intellectual burden on the engineering designers: For those decisions where the designers have room for choice and creativity, they must negotiate their decisions in light of all applicable laws and regulations. Typically, convergence towards compliant designs is a costly, lengthy, and non monotonic process involving repeated cycles of testing and patching. Integrated support of Verification and Validation (V&V) of designs with respect to applicable standards and requirements by Computer Aided Design (CAD) systems is highly desirable. Such support would likely have a noticeable impact on product quality and process performance.

The case for making verification and validation an integral part of engineering design activities does not need to be made. The benefits of integrating V&V with design are commonly agreed upon and are seen as the flagship of mature engineering domains (Dijkstra, 1989). Electrical engineers, for example, generally use fool-proof design methods and standards, leading to artifacts that are correct by design. Software engineers, by contrast, generally still design their products using error-prone trial and error processes (Mili et al, 1995, Shaw, 1990). Historically, design support systems have placed little or no emphasis on V&V support. V&V activities are either not supported at all or supported as an after-thought using a library of simulation and testing routines. An increasing need for V&V support, added to recent technological advances and trends has been raising the expectation for an integrated support of verification and validation as a standard feature in next generation CAD systems.

Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

The need for V&V support by engineering designers is increasing because the level of complexity of the systems being designed is increasing, and the level of specialization of the training received by designers is increasing. In a traditional setting, mechanical engineers are given the responsibility to fully design mechanical parts. Such mechanical engineers would have the adequate knowledge and skills to complete the design of the parts, verifying and validating them —mentally or otherwise— as they design them. This is no longer a typical design situation. In a more realistic setting, the person assigned the design task is not a mechanical engineer, but a technician trained in using a specific CAD system with possibly on the job experience designing similar parts. Also, typically, the part being designed is not a simple self-contained part that can be validated in isolation, but more likely, a component of a larger complex system. Furthermore, rather than being purely mechanical, the part is more likely to cross the boundaries of multiple engineering domains. The recently coined term mechatronics reflects this trend of disappearing firm boundaries between mechanical systems and electronic systems for example. In addition, partly because the part is a component of a larger system, its validity involves considerations beyond its functionality and manufacturability. Other typical considerations include assemblability, maintainability, reusability, recyclability, safety of the process and the product, and so forth. All of these factors preclude any single individual from being sufficiently knowledgeable and up to date in all of these specialized areas to take on the responsibility to design parts valid in all respects. Verification and validation cannot be made the responsibility of any single individual. System support is necessary to assure quality products and efficient processes.

From the technological side, two key resources are needed to enable an integrated V&V support by CAD systems. The first resource is the knowledge related to the correctness and validity of designs. Such knowledge must be collected, encoded, and managed. Knowledge collection identifies relevant knowledge. The encoding of the knowledge is necessary to make it usable by the CAD systems. Once collected and encoded, the knowledge needs to be managed to establish and preserve such qualities as currency, accuracy, completeness, and consistency. In engineering domains, knowledge collection is the simplest of the three. Universal (textbook) engineering knowledge is generally available, relatively well documented, and relatively formalized. In automotive engineering for example, the Society of Automotive Engineers documents thousands of standards and best practices related to all aspects of automotive engineering. These standards and best practices are published in a handbook of more than 4,000 pages updated yearly. In addition to this universal knowledge, the recent business trend recognizing corporate knowledge as a critical corporate asset (Huang 1998,O'Leary 1998, Taylor 1998) has led a number of companies to initiate knowledge collection and documentation efforts. Many engineering corporations have followed this trend and created engineering repositories documenting their inhouse know-how and best practices. These documents, once completed are invaluable resources begging to be used. A natural use of these documented standards and best practices consists of integrating them within the CAD systems to allow their systematic enforcement during design. This leads us to the second technological resource needed to make the integration of V&V within design a reality, namely, a design system that is *able* to use the knowledge available. A major hurdle with current commercial CAD systems is the wide semantic gap between the objects that they manipulate (points, lines and curves), and the concepts and features that are subject of the standards and regulations (steering wheel, driver's field of view). This situation is changing. The current trend in CAD research and CAD systems is to raise the level of abstraction of CAD objects and move from purely geometric concepts to more domain specific concepts and features (e.g. see (Anantha et al, 1996, Chu and Gadh 1996, Gero and Maher 1997, Reinschmidt 1994, Umeda and Tomyama 1997, Wong and Sriram 1993)). Some of the commonly used CAD systems (e.g. Catia, the Dassault systemes CAD system used notably at Boeing, DaimlerChrysler, and Lockheed; and Unigraphics, the UGSolutions CAD system used at General Motors) are undergoing major changes aiming at making them more flexible, more domain customizable, and more "knowledge oriented".

In sum, V&V support requires the collection, encoding, and management of domain knowledge, and the availability of a design support system able to use the encoded knowledge. In this position paper, we focus our attention on the encoding and management aspects of the knowledge. We present here our position concerning the representation, management, and use of domain knowledge in the context of engineering design V&V support.

Position

We articulate our position around three motivators: knowledge independence, knowledge integrity, and policy independence.

Representation: Support for knowledge independence

There are two competing requirements on the representation of the domain constraints reflecting its two categories of users: the domain experts who author and maintain these constraints, and the design support system whose task is to monitor and enforce compliance with these constraints. Experts conceive of the constraints as descriptive statements of valid designs. As authors of these normative constraints, they should be able to encode them, review them, and update them with ease. This dictates a declarative representation. The second user of the design knowledge is the CAD/CAM's V&V enforcement component. This component needs a set of operational triggers specifying the design activities that need to be monitored, the conditions to be checked, and the preventive and corrective actions to be taken. This calls for an operational representation.

Rather opting for one single representation and sacrificing one activity over another, we take the position that the two views need to co-exist. Domain knowledge is seen as a set of specifications of design objects and needs to be stored explicitly as such in a normative knowledge base. These design objects specifications' are represented by OO classes associated with sets of constraints. Enforcement and monitoring knowledge on the other hand is a set of operational directives derived from the domain knowledge as well as from other "policy"- and efficiency- motivated considerations. The normative and the operational repositories must be distinguished, but coordinated.

In sum, domain experts access and use a normative knowledge base. The representation used for this base promotes readability and maintainability. The design support system uses an operational knowledge base. The knowledge in this base represents the synthesis of various domains' knowledge bases as well as a an expression of preferences, tradeoffs, and priorities. An intelligent interface system is needed to synthesize the operational knowledge from various sources and express it as a set of efficient directives. This architecture promotes the independence of the domain knowledge from the processes enforcing it.

Management: Support for knowledge integrity

Knowledge integrity is the property by which the knowledge stored is a true reflection of the domain it is meant to represent. Knowledge integrity cannot be formally proved. It can only be painstakingly established through the careful crafting and review of the knowledgebase. Such care and close examination typically takes place upon creation of the knowledgebase. As the knowledgebase evolves and grows with the domain, there is a high risk that the integrity be compromised. This risk is especially high when the domain is dynamic —as is the case in engineering, and the number of expert contributors is high —as is likely to occur with employee turnover. The degradation of the coherence and integrity of knowledge bases with time is one of the major impediments to their widespread use. We take the position that integrity must be *promoted* by the representation chosen, and systematically *monitored* by the system.

The knowledge representation promotes knowledge integrity by minimizing redundancy, and explicitly documenting dependencies. We define and formalize a set of class inter-dependencies that occur commonly in design. We define a set of generic integrity constraints based on these relationships and similar to the normal forms defined for relational databases. We define a set of heuristics that can be used to monitor integrity. Details can be found in (Mili, 2000).

Use: Support for policy independence

The normative knowledgebase specifies the qualities that a designed product should ideally have. The operational knowledgebase specifies when to check specific qualities, and what actions to take if the qualities are not met. We discuss in turn the timing (when to check) and the response (what to do).

Qualities can potentially be monitored at every decision that may impact them. Such extreme approach is highly informative to the designer but would generally be perceived as too intrusive. Validation can also be activated at predetermined checkpoints in the design or upon request by the user. The checkpoints can be more or less spaced depending on the needs. The level and frequency of intervention of the system can be tuned to the level of criticality of the constraints and to the level of expertise of the user.

In a design setting, motivations behind constraints cover a variety of concerns ranging from feasibility, to safety, to cost. Also, different constraints are mandated by different bodies ranging from governmental agencies, to professional standardization committees, to in-house best practice committees. Different constraints may also have different levels of tolerance for violation. The action taken in response to a constraint violation needs to be attuned to the criticality of the constraint, its tolerance level, and the tradeoffs available between competing constraints. Preferences and priorities are used to help negotiate tradeoffs. In a low budget situation for example, cost constraints may be given priority over comfort constraints. If a luxury item is being designed on the other hand, cost considerations maybe compromised in favor of styling constraints. This variability illustrates policy independence: different policies can be used with the same domain constraints.

Summary, Conclusion

In this paper, we discussed issues relating to the representation and management of domain knowledge. In particular, we have focussed on three properties: Knowledge independence, knowledge integrity, and policy independence. Separating normative constraints from policies and priorities provides for independence and flexibility. We also highlight the need for system definition and monitoring of knowledge integrity.

Acknowledgement

Work on this project has been supported by the DaimlerChrysler Corporation and by the Michigan Research Excellence Fund.

References

R. Anantha, G.A. Kramer, and R.H. Crawford. Assembly modeling by geometric constraint satisfaction. *Computer-Aided Design*, 28(9):707–722, 1996.

Ch-Ch. P. Chu and R. Gadh. Feature-based approach for set-up minimization of process design from product design. *Computer-Aided Design*, 28(5):321–332, 1996.

E.W. Dijkstra. On the cruelty of really teaching computer science. *CACM*, 32(12):1,398–1,404, Dec. 1989.

J. Gero and M.L. Maher. A framework for research in design computing. In *Proceedings of ECAADE*, 1997.

K.-T. Huang. Capitalizing on intellectual assets. *IBM Systems Journal*, 37(4):570–583, 1998.

F. Mili. *Knowledge Architecture for Engineering Design Support.* Oakland University technical report, 2000.

H. Mili, F. Mili, and A. Mili. Reusing software: Issues and research directions. *IEEE TSE*, 21(6):528–561, June 1995.

D. E. O'Leary. Entreprise knowledge management. *IEEE Computer*, 31(3):54–61, March 1998.

K.A. Reinschmidt and G.A. Finn. Smarter computer-aided design. *IEEE Expert*, pages 50–55, 1994.

M. Shaw. Prospects for an engineering discipline of software. *Software*, 7(6):10–26, Oct. 1990.

Gerald H. Taylor. Knowledge companies. In William E. Halal, editor, *The infinite resource*, pages 97–110. Jossey-Bass Publishers, 1998.

Y. Umeda and T. Tomiyama. Functional reasoning in design. *IEEE Expert*, pages 42–48, March-April 1997.

A. Wong and D. Sriram. Shared: An information model for cooperative product development. *Research in engineering design*, (5):21–39, 1993.