Unified Representation systems for different levels of abstraction

Arthur B. Baskin
Senior Analyst, Intelligent Information Technologies, Corp.
123 W. Main St., Urbana, Illinois 61801, U.S.A.
Telephone: 217-337-7058; Fax: 217-337-6928
E-mail: abaskin@msn.com

Stephen C-Y. Lu
David Packard Professor of Manufacturing Engineering
University of Southern California, Denney Research Building, Room 101, Los Angeles, CA 90089-1111, U.S.A.
Telephone: 213-740-9616; Fax: 213-740-6668
E-mail: sclu@usc.edu

Abstract

Our unified representation system for organising and recording engineering design results brings together formalisms from mechanical engineering design and software engineering. It specifically identifies two orthogonal axes of abstraction and explains many problems in engineering design formalisms, which lack this separation. The formalism supports qualitative and quantitative measures of design quality that can be used to evaluate alternative design structures. The formalism does not prescribe a specific design methodology, but the representation does provide bias toward more formal and reproducible design methods. Although the formalism is grounded in object-oriented information modelling, its application to engineering design problems is much broader than software development.

Keywords

Knowledge representation, Axiomatic Design, modelling, structured methodologies, engineering design, Engineering as Collaborative Negotiation

---

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35357-9_22

A. B. Baskin et al. (eds.), *Cooperative Knowledge Processing for Engineering Design*

1 INTRODUCTION

Engineering design can be thought of as a problem of managing complexity. A product should appear simple to the consumer and simple to assemble or. Thus, engineers must match the complexity inherent in a product or service with a design while avoiding unnecessary complexity.

Engineers traditionally manage complexity by decomposing problems into smaller, less complex, component problems that can, in turn, be assembled to form an overall solution. Complex system-level design problems can only be solved by decomposing them into sub-systems that generally have interacting constraints. Forming separate sub-problems enables work to proceed in separate areas in parallel, but gives rise to conflict and rework because differing approaches to the overall problem may be implicit in each of the sub-systems. The component models, provided by these parallel design activities, frequently employ differing specialised vocabularies and ways of structuring the solution. For these reasons, integrating the component solutions and solving the resulting conflicts frequently consumes much of the productivity that resulted from the concurrent design activity. Such a collaborative design framework is illustrated in figure 1 below.

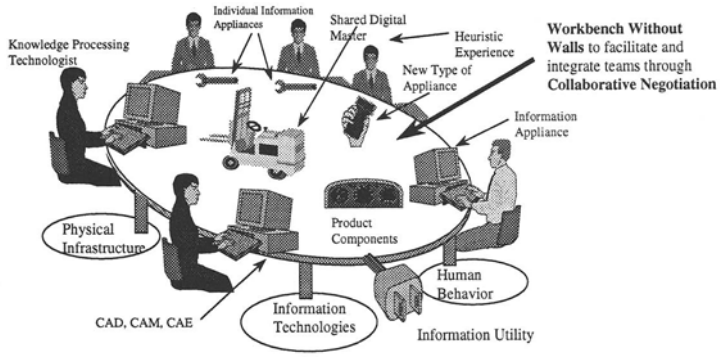

Figure 1: A group of designers using a shared electronic workbench to develop a solution of a system-level design problem.

One might imagine integrating these diverse design vocabularies and different localised optimisation criteria by imposing a unified vocabulary and prescriptive methodology on all of the component design activities. Unfortunately, this approach re-introduces all of the original complexity of the presenting problem into each component. The solution lies in having a unified representation system, which promotes the convergence of the group, while allowing for diversity.

2 TWO ORTHOGONAL AXES OF DECOMPOSITION

Part-whole decomposition is a way to break an overall problem into manageable components, which can be separated from the decomposition of the problem across product life cycle stages. At first, this separation may appear artificial because both axes proceed from abstract properties of the presenting problem toward more concrete aspects of the problem and its implementation. The separation of more concrete aspects of the problem from concrete aspects of the implementation of the solution is central to our unified representational framework.

Following the pattern, established by Professor Suh in Axiomatic Design (Suh 1990), we identify four key stages in the decomposition across product life cycles. (Identification of additional stages is a simple extension of the formalism.) Figure 2 illustrates the progression from problem toward implementation.

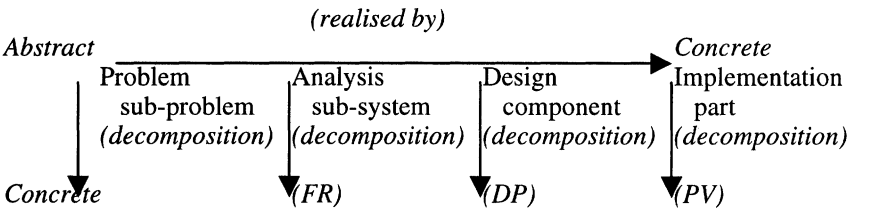

Figure 2: The relationship between abstraction across product stages and within each stage. Separating the two orthogonal abstraction processes into *decomposition* and *realised by* occurs at each stage in the product life cycle.

In Axiomatic Design, the last three stages are identified as Functional Requirements (FR), Design Parameters (DP), and Process Variables (PV) respectively. The important part of this figure is the illustration that there are two orthogonal forms of abstraction at each product stage. We will refer to the abstraction within a stage as "decomposition" and the orthogonal abstraction operator that maps between product stages as "realised by."

2.1 Separating abstractions provides reproducibility of placement

We wish to bias the actions of a group of designers from different engineering perspectives so that they will share as much common structure in the results as practical while avoiding a unified and prescriptive approach. In the figure above, the two different axes of abstraction will naturally lead to an arrangement of design results within a two dimensional lattice. Aspects of the overall problem will occur in the upper left corner and detailed implementation (fabrication) issues will occur in the lower right corner. Thus, co-operating designers will know where to attach their intermediate results and where to look for the results of others.

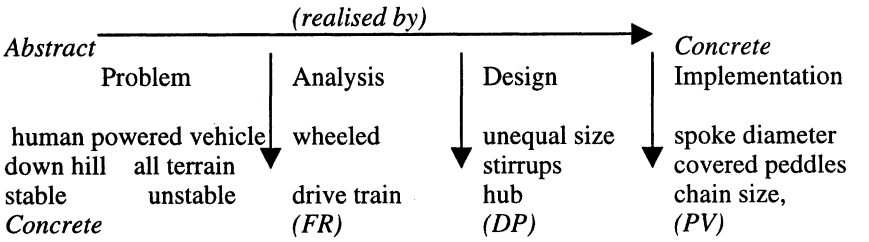

Figure 3: A simple model of the problem of designing a human powered vehicle, which shows the relative placement of terms used in the design.

A concrete example may help illustrate the point. Figure 3, shows how a number of the terms in the design of a human powered vehicle would be arrayed on our two dimensional abstraction grid. The upper terms can be specialised by the terms below them and the terms to the right are used to realise the terms on the left.

2.2 Objects make placement more formal

Thus far, we have used an informal notion of placement of terms within the two dimensional abstraction grid by separating the terms according to the product stage which best typifies the term and the level of detail of the term within the given stage. We now use an object-oriented information modelling formalism to make these notions more precise. We choose an object-oriented formalism because it can be made as mathematically precise as we wish and can be extended with specialised domain relationships.

For the purposes of this paper, we use a simplified version of the OMT modelling technique developed by (Rumbaugh 1991). In this simple model, we associate nouns with objects and verbs with methods on objects. In addition to objects, we utilise several predefined relationships having the symbolic representation shown in the figure below.

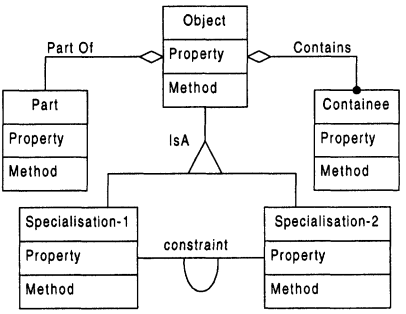

Figure 4: Object modelling notation showing the "part-whole," "container-containee," "constraint," and "IsA" relationships among object. The presence of relationships indicates coupling among the objects.

We use these types of relationships between objects to model aspects of the design at each stage. Thus, we say that a "table object" has a table top and four separate legs – all part of the table because these parts are essential to the function of a table. On the other hand, a "room object" can contain a collection of people of any size without changing the fact that it is a room. Similarly, a "conference room" or a "bathroom" are both specialised rooms and we might wish to record the domain constraint that it is a good idea to have a bathroom "near" a conference room.

2.3 Four object models for four separate purposes

We continue our discussion of where to place a given concept during the construction of the design by considering the function of each of the four models in greater detail. We expect that all of our participating designers will use the same set of rules for placing a noun in a given model and use the same definitions of the object relationships between them. In this situation, there should be substantial agreement among the participants about the overall structure of the design model without imposing a single vocabulary and design methodology for all perspectives.

The objects, which are identified in the problem decomposition, should all represent entities or concepts occurring in the problem domain. Thus, these domain terms serve to define the nature of the problem area and how to survey the problem in a structured way. Only nouns that occur directly in the problem domain should be used for objects in the problem model. The problem model should delimit the problem but should not specify aspects of the solution unless they are dictated by the nature of the problem itself.

The objects in the decomposition of the analysis model should be used to define "what" will be done in response to the problem, but should not specify "how" it is to be done. For example, in our bicycle example, the analysis model might specify "gears" to be part of the "drive train" without specifying the design choice between gears internal or external to the hub of the rear wheel.

The objects in the design model should represent abstract solution strategies rather than specific settings. For example, the choice between casting or machining a specific part is a design choice, while the specific numerical control program for a milling machine provides a detailed implementation model.

2.4 Relationships record context, show alternatives, and promote reuse

The problem of design reuse is quite difficult and has many aspects. Part of the problem of reusing an existing design comes from the difficulty in locating an existing design and then reconfiguring the design to the presenting situation. When we consider system-level design problems solved by groups of co-operating designers, we see that this problem contains a version of the design reuse problem. When two designers with different perspectives on the problem come together to resolve conflicts or solve interacting constraints, they must be able to understand the basic structure of each other's model. This model sharing activity is little changed if the model was developed some time ago and is only being shared today

for the purpose of reuse. Navigating the model, understanding the specialisation implicit in the design result, and tailoring it to a different use are much the same problems for design reuse and collaborating perspectives.

The modelling formalism we have discussed thus far, naturally records the path of specialisation which goes on during model construction. The decompositions of each of the four models and the realised by mappings record the progress of the specialisation of the problem from a general problem statement in the top left corner to a detailed implementation model in the lower right corner. Maintaining these links explicitly throughout the life of the modelling process is vital to understanding the model today and reusing it tomorrow.

It is possible to record, in the model, objects which are not strictly part of the solution. For example, in the problem model, we might show that there are human powered vehicles and engine powered vehicles using an IsA relationship which branches from the abstract notion of vehicle to each more specialised vehicle. This form of "branched IsA" shows that there are alternative problem specialisations from which we can choose. We record our choice by drawing a "realised by" arc from only one of these specialised problems. In this way, we record the problem context for the problem we are solving.

In a similar way, "branched realised by" denotes that there are two or more alternative ways to realise a given requirement. As we accumulate design experience in a given area, we can record each of the realised by arcs which we explore from a given point. This "map" records the alternatives explored and suggests the alternative ways that the problem can be specialised. Taken together, the "branched IsA" and the "branched realised by" arcs provide just the sort of road map that we require. We can use the map to understand how a problem has been specialised for an existing design solution and how to specialise it differently when solving conflicts in concurrent engineering or reusing it at a later date.

3 APPLYING QUANTITATIVE MEASURES TO DESIGN MODELS

Perhaps the most important property to measure about a model is the degree to which the model "appropriately" matches the complete system being modelled. In terms of our formalism, this measure corresponds to determining whether or not the object identities and relationships have been properly chosen. For the implicit realised by arcs from the real world into the problem model, it is impossible to evaluate the quality. Thus, the fidelity of the problem model cannot be evaluated by our metrics, which operate only on the model. We can, however, measure the degree to which the problem model is properly reflected in analysis, design, and implementation models. We can also apply measures of quality within each individual model as well.

A number of evaluation metrics have been proposed for object-oriented software development (Henderson-Sellers 1996). A number of these can be adapted to the more general problem of object-oriented information modelling but this extension is beyond the scope of this paper. In the remainder of this section, we will explore

a measure of mechanical engineering design quality, generalised from Axiomatic Design, which applies equally well in this object-oriented design context.

3.1 Patterns of coupling indicate robustness

We can consider that two objects are "coupled" if and only if there is a relationship existing between the objects. For this discussion, we ignore the role of the type of relationship on the strength and detailed nature of the coupling.

Before considering a quantitative measure of robustness, we explore the notion qualitatively. There can be many ways to define robustness. The definition that we favour is that if either the requirements or detailed implementation options are changed, the ripple effects through the design are damped. If either change triggers redesign efforts out of proportion to the size of the original change then the original design was not robust. What this means is that if the pattern of coupling in our solution matches the pattern of coupling in the problem, then the changes in the solution can parallel those in the domain and the solution will be robust.

3.2 Quantifying Robustness Using Coupling Matrices

We will now proceed to quantify this notion of robustness as "matching the pattern of coupling in the domain." We can summarise this constraint, between the problem (P) and the solution (S), using the following matrix equation:

$$| {}_{P}C | = | {}_{S}C | \qquad (1)$$

where the coupling terms ${}_{P}C_{I,J}$ are non-zero if there is coupling (an object relationship) between elements I and J of the problem model and zero otherwise. Similarly, the coupling terms ${}_{S}C_{I,J}$ are non-zero if there is coupling (an object relationship) between solution elements I and J and zero otherwise. Note, if terms ${}_{P}C_{I,J}$ are non-zero and the corresponding coupling terms ${}_{S}C_{I,J}$ are zero, then the solution is less coupled than the problem, which causes no harm. If the converse is true, then the problem elements can change more freely than the solution, and the solution is, therefore, not robust under changing requirements.

If we expand (1) to include the separate analysis, design, and implementation models, we can decompose the coupling within the solution into its components:

$$| {}_{P}C | = | {}_{S}C | = | {}_{PA}C | * | {}_{AD}C | * | {}_{DI}C | \qquad (2)$$

where the new terms correspond to the solution coupling arising from analysis, design and implementation respectively. Notice that the matrix algebra for coupling terms mirrors the transitivity of coupling in the actual models. An overly coupled implementation will effectively lead to a solution that is coupled even if the analysis and design models are not.

We now separate the coupling which results from the realised by arcs between models from the coupling within a given model. Each of the component terms in equation (2) can now be expanded into its two components:

$$|{}_{P}C| = |{}_{S}C| = |{}_{RA}C| * |{}_{A}C| * |{}_{RD}C| * |{}_{D}C| * |{}_{RI}C| * |{}_{I}C| \quad (2)$$

where we have shown the coupling matrices for the realised by mappings between the models together with the coupling matrices within the models for analysis, design, and implementation respectively. (This separation is equivalent to the notions of inter module and intra-module connectivity in software engineering.)

In the equations above, the coupling terms are, to first approximation, 0 if there is no connection between the two referenced elements and 1 if there is. Note that most relationships can be unidirectional and therefore the matrices will generally not be symmetric about the diagonal (which always contains ones for the coupling within a model). We now turn to the mapping between models embodied in the "realised by" arcs. Like the arcs making up the decomposition within a model, these arcs can also define a pattern of coupling.

If we use (2) to compare the pattern of coupling in the problem to that of the solution and we discover a coupling term in the solution, which is larger than the corresponding term in the problem, then there is a potential problem. According to the problem model, it is possible to vary two or more requirements separately, but the solution model is not able to isolate that variation. Thus, functionality in the solution unrelated to the requirement change must also be adjusted. Thus, the requirements change are "amplified" rather than "damped" as we require. Using the component matrices in equation (2), we can identify the source of the excess coupling as originating from one of the realised by mappings or one of the analysis, design, or implementation models. Thus, we can target the improvement of the model using this approach. The most robust design is the most likely to be well behaved when our co-operating designers come together for a design review and attempt to reconcile inconsistencies in their competing partial solutions.

3.3 Applying coupling measure at different levels of abstraction

As we have seen above, coupling matrices can be used to compare the amount of coupling in the presenting problem and the way in which the solution matches this pattern. Because of the nature of matrix products, we can combine the coupling matrices for analysis, design, implementation and their respective realised by mappings into a single combined matrix which summarises the pattern of coupling in the overall solution.

In the event that the solution is more coupled than the presenting problem, we can use the individual decomposition and mapping matrices to detect where the additional coupling is being introduced. (Once a non-zero coupling term has been introduced, it will propagate through the matrices in the same way that the functional coupling propagates through the design.) In this way we can apply our robustness metric to the overall solution and to its major component parts.

Thus far, we have explored a relative scale for robustness based on comparing the pattern of coupling in the solution to that of the problem. As we apply the metric to more detailed situations, we can develop a restricted metric (which corresponds more directly to the metric used in Axiomatic Design).

If we assume the worst case presenting problem, each element of the problem will be able to vary independently. In this situation, the coupling among the problem elements will produce the identity matrix. In general, we will not be able to force the product of the mapping matrix for realised by arcs and the coupling with the various models to produce the identity matrix. We now relax our earlier matching restriction to require that the matrix be lower triangular (normal matrix reordering can be used to collect the zeros above the diagonal).

Solutions whose coupling matrices are at least lower triangular are still robust against changes in requirements. A triangular coupling matrix implies that there is a single degree of freedom associated with one element of the solution that forms the context for all remaining solution elements. Once this element of the solution has be set, the next most heavily coupled element can be set. Proceeding in this way, the elements of the solution can be set in a single pass through the problem. In the event that the solution is not lower triangular, it means that there is a cycle in among the couplings. A solution for this cycle must be identified as a single unit and cannot be found in a convergent manner. Thus, cycles correspond to non-triangular matrices and non-robust solutions.

3.4 Automating the evaluation of design quality measures

The coupling matrices necessary to evaluate relative coupling between problem and solution can be evaluated automatically while design models are constructed. Complete coupling matrices require that the parameter lists of each object's method specification must also be examined. Method parameters can provide additional coupling beyond that which is apparent in the topology of the models.

A number of software engineering frameworks routinely collect such data about object-oriented software models defined with the tool's framework. We are currently exploring the extension of such frameworks to include our robustness measures in addition to more traditional object quality measures.

4 RECONCILING PERSPECTIVES AND REUSING PATTERNS

We are now ready to return to the problem of providing a generalised representation for design results that is usable by designers from different perspectives. We will add a third dimension to our modelling framework: reuse of design patterns. As we pointed out initially, the challenge for integrating the work of designers from differing perspectives is to support communication and co-ordination among the designers without imposing a single unified vocabulary or prescriptive design method.

We identify a design perspective with a subset of the overall models. The subset may be contained entirely within a single product stage or may cut across stages as required by the engineering perspective. Different engineering perspectives may be substantially disjoint in their attention to elements of all four of the individual decomposition models. In addition, each perspective may only treat a subset of the realised by arcs between the model elements treated by the perspective.

In our framework, perspectives are not permitted to be entirely disjoint. Each perspective must be a specialisation (usually with extension) of a base pattern of structure, which is common to all perspectives. Each perspective can specialise only a portion of the global pattern, but this pattern enforces a minimal connectivity among the perspectives.

Because each perspective maintains its knowledge about the patterns of coupling independently, even if summary results are shared, it is possible for the group to define coupled solutions without knowing it because the coupling matrices are never assembled across all perspectives. In our framework, the developers of each perspective are permitted to work with their own subset of the overall model. The use of a single framework and a single store for the emerging perspectives allows the detection of undesirable patterns of structure before they lead to rework during design integration or brittle response to changing requirements.

The basic patterns from which each perspective is specialised can be used to collect normative experience in each of the product stages. In the first stage, we record proven ways to structure families of problems and we can frequently map these problem elements to meaningful analysis elements. In a similar way, analysis patterns convey the experience in how to approach the solution to well understood problems. We represent this specialisation of normative patterns at each stage of the product life cycle as a third orthogonal axis in our modelling framework. This axis is used to record the abstraction of experience with families of problems and solutions and recourse to these libraries of patterns helps to unify the competing design perspectives.

5 DISCUSSION AND SUMMARY

In this paper, we have presented a formalism for recording engineering design results which emphasises the separation of traditional sub-system decomposition into two orthogonal abstraction hierarchies: one spanning product life cycle stages and one decomposing a single stage. We have shown that some patterns of connection among objects in this two dimensional model structure correspond to robust structures, which damp ripple effects due to change in requirements or conflict resolution among co-operating designers. These patterns can be automatically evaluated and used to localise undesirable coupling within the overall framework or evaluate alternative designs.

The nature of the orthogonal separation of axes of abstraction provides a natural localising and organising framework, which can be used to guide a group of engineers approaching the problem from different perspectives. More general patterns can be included in pattern libraries and specialised as part of a program of design reuse. Finally, maintaining a fully connected design model within the framework provides documentation of the alternatives available and explored along with the rationale for each implementation element.

6 REFERENCES

Suh, N. P. (1990) *The Principles of Design*, Oxford University Press, New York.
Rumbaugh, J., et. al. (1991) *Object-oriented Modeling and Design*, Prentice-Hall.
Henderson-Sellers, B. (1996) Object-oriented metrics: measures of complexity, Prentice-Hall.

7 BIOGRAPHIES

Arthur **Baskin** has been involved in applied artificial intelligence and decision support systems research for the past fifteen years. After receiving his Ph.D. in computer science from the University of Illinois in 1979, he has developed information systems in medicine, civil engineering, mechanical engineering, and computer assisted instruction. He is now a senior analyst with Intelligent Information Technologies working in the areas of object software methodology and support systems for engineering maintenance management.

Dr. Stephen **Lu** holds the David Packard Chair in Manufacturing Engineering at the School of Engineering at the University of Southern California (USC). Besides leading the school-wide design and manufacturing systems research program, he is also the founding director of the IMPACT Research Laboratory and the Asia Pacific Institute at USC. His current research interests are in the development of basic theories, decision models and computer tools to support engineering as collaborative negotiation. He has over 160 publications in the areas of engineering automation, and received many national and international awards for his technical contributions.