Towards A Framework For Concurrent Design

by

S. Talukdar, S. Fenves

EDRC 18-14-90^

TOWARDS A FRAMEWORK FOR CONCURRENT DESIGN

Sarosh N. Talukdar **Steven J. Fenves**

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213
41 2-268-8778

ABSTRACT

By a framework for concurrent design we mean: (1) formal ways of stating the problems of concurrent design, (2) visualization (conceptualization) aids to help devise strategies for solving these problems, and (3) implementation aids to help translate the strategies into working systems.

This paper begins by defining some terms, including "conflict," and "computational path." Next, concurrent design problems are formulated in these terms. Specifically, these problems are shown to be equivalent to finding computational paths that avoid or eleminate conflicts and connect given data-objects to desired data-objects. A class of graphs, called TAO graphs, is developed for visualizing such paths. Finally, a computational environment, called FORS, is described for implementing selected paths.

INTRODUCTION

The design of a complex artifact, such as a car, a bridge or **a** microelectronic chip, involves a large number of tasks each requiring a different sets of skills. Often, the groups to whom such tasks are assigned work on their assignments at different times and in different places. Nevertheless, their work must be coordinated to take into account the couplings among the tasks. Because of these couplings, decisions made in one task can affect some or all of the others. The decisions made in the early conceptual tasks are of particular concern. These decisions can have profound effects on all the tasks that come later. Often, these effects are deleterious, making it difficuJt or impossible to perform the later tasks well.

The concurrent design problem can be stated simply as follows: how can the propogation of deleterious effects from one task to another be reduced to tolerable levels? (Henceforth, we will refer to intolerable effects as conflicts or inconsistencies).

Proceedings of the Special Session for Concurrent Design, ASME Winter Annual Meeting, San Francisco, CA, December 11-14, 1989

Strategies for solving the concurrent design problem can be divided into two broad categories:

1. Preventive or look-ahead strategies which seek to anticipate conflicts and avoid them before they occur.
2. Corrective or feedback strategies which allow conflicts to occur and then use backtracking or iteration to eleminate them.

Corrective action is usually more time consuming and therefore, less desirable than preventive action.

Existing techniques for both prevention and correction tend to be people-intensive and cumbersome. Typically, experts from each of the major design areas are assembled into a group that oversees the design project from start to finish. This requires considerable committments from the experts and powerful mechanisms to transcend the communication barriers that inevitably arise among them.

To help automate prevention and correction processes we propose a visualization aid called a TAO graph. Nodes in this graph denote the declarative portions of design work while directed arcs denote the procedural portions.

We think of the declarative portions of a design as a set of data-objects each containing information on one view or aspect of the artifact being designed. A set of specifications, a set of sketches, a set of blueprints, a parts list, a manufacturing plan, and a prototype are some of the very many different aspects of a car.

Both the inputs and the outputs of design processes can be expressed in terms of aspects and a general form of the design problem is: given the values of certain input aspects and the types of the desired output aspects, find values for the output aspects that are consisten with the inputs and one another. For example, given the specifications of an automobile engine, find or develop blueprints and a manufacturing plan that are consistent with (meet) the specifications and are also consistent with each other.

The procedures by which design problems are solved can be divided into two categories: manual procedures (that are executed through humans) and automatic procedures (that are executed through computers). We visualize both types of procedures as operators that map the contents of one data-object into another (i.e., one aspect into another). Thus, we visualize design activity, both manual and automatic, as tracing paths through TAO graphs. These paths begin at given, input aspects and pass through operators and intermediate aspects to terminate at output aspects. For the activity to be successful, the output aspects must be consistent with one another and the input aspects.

By displaying the different computational paths that link input and output aspects, TAO graphs provide a way for both visualizing design problems and planning strategies for solving them. However, before they can be used, a number of terms such as aspect, operator and consistency, must be more precisely defined. We will suggest definitions for these terms in

the immediately following material. The remainder of the paper is devoted to a description of FORS, a programming environment for implementing strategies for solving concurrent design problems.

TERMINOLOGY

Aspects

Let Af, A2,—, be sets or spaces and let ajj, aj$_f$2,— be elements or points of Aj, such that a{j is the j-th instance of the i-th view or feature of an artifact. To illustrate, consider an artifact whose i-th feature is a resistive circuit that is known to have 10 or less nodes. Then a•, j$_f$ aj$_f$2.---. are all the different resistive configurations that are possible with 10 or less nodes and Aj = { a|j, *a\$_{i2}$*,~- }.

We will refer to the a's as aspects and to the A's as aspect-spaces or data-objects depending on whether we are discussing conceps or programming implementations.

Operators

An operator is a mapping from one aspect-space to another. Thus, the general form of an operator is:

$$°i,j : {}^{A}i \text{ "> } {}^{A}j \quad {}^{or} \quad {}^{a}j - Ojj(aj) \qquad (1)$$

where Aj is the input-aspect-space of the operator and Aj is its output-aspect-space.

A Taxonomy of Operators

Three dimensions are useful in classifying design operators. They are:

1. Degree of autonomy. This dimension can be used to divide operators into three categories:
   - -autonomous: operators that act largely on their own volition, for instance, most human designers,
   - -semiautonomous: operators that will accept commands but can also act on their own volition, for instance, intelligent programs that go looking for tasks when they are not busy with tasks that have been assigned to them,
   - -non autonomous: operators that do nothing until they are assigned a specific task, for instance, conventional simulation and analysis programs.
2. Effect on information content. This dimension deals with the relative information content and location of an operator's input and output aspects, a; and aj. Based on this dimension, operators are divided into four categories (Fig. 1):
   - -abstractors: aj has less detail than aj
   - -refiners: aj has more detail than aj
   - -translators: aj and aj are informationally equivalent
   - -modifiers: aj and aj are in the same aspect-space.

3. Function. This dimension deals with the specific type of transformation performed by an operator. A few of the categories along this dimension are:

Synthesizers : operators that transform specifications into design alternatives;

Analyzers and *simulators* : operators that evaluate the performance or calculate the responses of given alternatives to given stimuli;

Optimizers : operators that improve on design alternatives.

Consistency

Consider two typical aspects: a set of specifications and a set of blueprints. How can one test such aspects for consistency, that is, check to see if the blueprints meet the specifications? In many cases the only practical approach is to transform the blueprints into a physical prototype that is subjected to a number of laboratory tests. In other words, one must select operators to transform the specifications and blueprints into points in a common aspect-space (the space of laboratory experiments) and then check to see if the points are coincident or close. More formally, the ideas involved can be stated as follows:

Aspects a_i and a_j are consistent to degree b, if:

(1) there can be found widely accepted operators $O_{i,k}$ and $O_{j,k}$ that map a_i and a_j into a single space A_k,

(2) there exists a distance metric $\| \cdot \|$ for the space A_k, and

$$(3) \quad \| O_{j,k}(a_j) - O_{i,k}(a_i) \| \leq b \qquad (2)$$

The selection of the operators and the distance metric imparts a subjective quality to the test that is absent from purer fields, such as mathematical logic, where consistency can be defined objectively. Another difference is that in design it is more convenient to think of consistency as a continuum or fuzzy set rather than as a binary proposition.

Design Tasks

The general design task has the following form:

Given: (1) two aspect spaces, A_{in} and A_{out};
(2) a point (aspect) a_{in} in A_{in};
(3) a consistency threshold b and a test for checking this threshold;

Find: a point a_{out} in A_{out} such that a_{in} and a_{out} are consistent to degree b or better.

Conflicts

Let: AAout be the subset of A^j that contains aspects that are consistent with aj_n to degree b or better. We will say that there is a conflict between the task and its input if AA_{0Ut} is empty. In other words, conflicts are said to occur when there is no solution to a task that is consistent to the required degree with its given input. Such conflicts often occur as the result of the solutions selected for upstream tasks. For instance, the design of the structure of an artifact could make it impossible to manufacture. If allowed to persist, such conflicts result in degradations of quality and performanceanf their elemination is the objective of concurrent design.

Computational Paths

By a computational path we mean a partial ordering of operators for solving one or more tasks. To illustrate, consider a simple task that involves solving a pair of nonlinear algebraic equations:

$$x = f(x,y) \quad (3)$$
$$y - g(x,y) \quad (4)$$

Let x*, y* be a solution to these equations. One iterative algorithm that can be applied to search for this solution is:

(P1): $x_{n+1} = f(x_n,$ (5)

$n,m = 0,1,2,\text{™}$

$ym+1 - g(^{x}m . ym)$ (6)

Another algorithm is:

(P2): $^{x}n+1 - {}^{f}(^{x}n. y_{+})$ (7)

$n,m = 0.1.2,\text{™}$

$ym+1 - g(x_{+},y_{m})$ (8)

where x_n, y_m are the n-th and m-th approximations to x*, y*, and x+, y_+ are the latest available values of x_n, y_m. Let Of and Og be operators for evaluating functions f and g. The partial ordering of these operators corresponding to the two algorithms are shown in Figs. 2 and 3. The first algorithm requires the two operators to be invoked in lock step; every time Of is invoked Og must also be invoked. The second algorithm, however, allows the invokations of Of and Og to proceed independently (that is, asynchronously). Neither algorithm enjoys a clear superiority over the other; in some cases the first is preferable, in others, the second (Talukdar et al., 1983). Infact, even for this very simple task, there are other partial orderings of operators, each with its own merits, and the number of these orderings increases dramatically if one considers operators besides the two listed above. As we shall see, the problem of designing a good

partial ordering of operators for solving this simple task, is in microcosm, the problem of concurrent design.

Design Systems

A design system consists of decision making agents (people and programs) supported by passive resources such as technology bases and laboratories. The agents tend to be arranged in complex hierarchies, that is, structures with multiple levels in which an agent can report to two or more agents in the level above. Some of the agents serve as operators, the others serve as managers. Managers are responsible for setting goals, decomposing large design tasks into smaller tasks and assigning these tasks to operators. As such, the operators occupy the lowest level of the organizaion.

The purpose of a design systems is to select and implement computational paths for performing given design tasks.

TAO GRAPHS

A TAO graph (the "TAO" stands for Tao or path in Chinese, the "A" for aspect-space, and the "O" for operator) is an aid for visualizing the capabilities of the lowest level in the organizational structure of a design system, that is, the level in which the operators reside.

A TAO graph is an *and/or* graph whose nodes represent aspect-spaces and whose arcs represent operators. As such, a TAO graph depicts all the paths through a given set of aspect-spaces that are made possible by a given set of operators. Since both aspect-spaces and operators can be aggregated and disaggregated, this perspective can be widened to view the paths for an entire project or narrowed to view an individual task in any desired degree of detail.

The purposes of TAO graphs are (1) to help view available paths and select the best from among them, thereby, aiding in the design of design projects, and (2) to compare the best available paths with desired paths, and thereby, determine how to upgrade an existing design system or design a new design system.

An Example

The Integrated Building Design Environment (IBDE) is a prototype system for the design and construction of high rise, speculative office buildings. IBDE is being assembled to serve as a testbed for the exploration of three sets of issues: (1) the information, control and communication needs of the diverse agents required for the design and construction of a building; (2) the applicability and role of generic tools and design environments in the domain of architectural and civil engineering design; and (3) concurrent design.

IBDE is being developed in two phases. The first phase, which has been completed, involves the integration of seven automatic operators that cover the architectural, structural and foundation design, and the construction planning of high rise

buildings. These operators are implemented as knowledge-based systems (KBS's) to facilitate rapid development and modification. While some of the current operators are too limited to serve as models of practical programs, they are comprehensive enough to serve as surrogate experts in exploring the types of design representations and communication mechanisms needed for the concurrent design of buildings and their construction process..

This first phase of IBDE uses the following aspect-spaces:

- A1: building owner's targets (area and cost), objectives and constraints
- A2: building massing, functional assignments of spaces, vertical circulation area
- A3: structural grid: a 3D grid of bays and stories where structural elements can be placed
- A4: optimal spatial layout of elevators, stairs, restrooms etc. occupying the building's core
- A5: structural system selection
- A6: structural layout and approximate forces acting on structural components
- A7: design of structural components
- A8: design of foundation components
- A9: construction project activities, sequence, costs, durations

A TAO graph of these aspect-spaces and the current complement of operators is shown in Fig. 4. In making available an automatic path that leads all the way from A1, the building's specifications, through several intermediate spaces to a construction plan, IBDE provides a larger scale of design tool integration than any other building design system that we know about. However, notice that there is one and only one path. As we will point out in the next section, concurrent design requires multiple paths and in the second phase of the project, which is now underway, the IBDE team is adding an appropriate set of additional paths.

CONCURRENT DESIGN

Over the last few years, the terms "concurrent design" and "simultaneous engineering," have come to be used for technologies that strive for high quality in a design from several different and often conflicting points of view, such as those taken by the marketers, manufacturers, users, and maintainers of an artifact. Each such group has its own stringent set of specifications that must be met in order for it to consider the

artifact to be of high quality. In our terminology this problem can be posed as follows:

Given: a number of design tasks in which each task can have several input aspects, some that are independently specified, others that are computed by previous tasks;

Find: (1) a path to perform each task;
(2) a means for integrating these individual task-paths into a composite path such that all the input and output aspects are consistent (conflict free).

Thus, in essence, the concurrent design problem is one of conflict-free, path integration. As was pointed out earlier, the two basic types of approaches to achieving this sort of integration are preventive and corrective. In the former, conflicts are anticipated and avoided; in the latter, conflicts are allowed to occur and then reduced to tolerable levels through backtracking and iteration. When it is done well, prevention tends to be faster. The difficulties are in the breadth of knowledge required and in the extended horizons over which predictions must be made. To illustrate, consider the IBDE case (Fig. 4). In making massing and space allocation decisions (A->A2), knowledge of all the subsequent design tasks must be brought to bear to predict impacts and potential conflicts to the end of construction planning (A8->A9). As has been pointed out, existing approaches to solving the prevention problem tend to be manual and distributed (a team of experts with a wide enough range of knowledge is assembled to predict and resolve conflicts). Our research has focussed on automating these manual processes. Specifically, we have been investigating the substititution of a team of expert systems, called *critics*, for some or all of the team of human experts. Each critic consists of a simulation, analysis or forecasting program sandwiched between an intelligent pre- and post-processor. The pre-processor is required to:

1. understand the representations used by all the synthesis tasks that might require the services of the critic. For instance, the critic for predicting conflicts with the construction planning task in IBDE (A8->A9 in Fig. 4) must understand the representations used by all the preceeding synthesis operations.
2. monitor the progress of these synthesis operations and activate its simulation or forecasting program whenever necessary. For instance, if decisions being made in the A1->A2 operation (Fig. 4) are likely to have effects on construction planning, then the associated critic should activate itself and predict the extent of these effects.

The purpose of the post-processor is to summarize and explain the predicted results.

Thus, a critic is an autonomous operator that contains an analysis or forecasting program augmented with self activation and self explanation capabilities whose collective purpose is the automatic identification of potential downstream conflicts. Once such conflicts have been identified they must be resolved. We are

in the early stages of developing automatic procedures for conflict resolution.

Ongoing Work

In IBDE, work is now underway to enrich the computational path throgh the addition of a number of critics and two conflict resolvers, one at the level of spatial conflicts between aspects A3, A4, A5, and the other at the level of construction cost and time conflicts between the owner's targets (A1) and the system's projections (A9). General rules or solutions arising from the implementation of these conflict resoivers may eventually be incorporated into a generic and automatic look-ahead strategy for conflict elemination.

Besides IBDE, we have been developing platforms for invesigating critics in two other areas: the real time control of electric power systems (Stoa et al., 1989) and the design of automobile parts (Sapossnek et a!., 1989). Work in the real time area is at about the same stage as IBDE, that is critics are just being introduced. However, in the area of automobile parts a number of prototype critics have been completed and from the results of preliminary testing, seem capable of producing considerable improvements in both design time and quality.

FORS

Notation

In this section we will use the terms data-object and tool-object to mean programmed versions of aspect-spaces and operators. Data- and tool-objects are obtained by adding front ends to new or existing programs to provide them with object-like capabiHties. The principal function of a data-object is to store aspects; of a tool-object, to transform aspects.

Philosophy

Earlier sections of this paper have argued that (1) problems of concurrent design are equivalent to problems of designing integrated, conflict free, computational paths that connect given aspects to desired aspects; and (2) TAO graphs are useful in visualizing and designing such paths. In this section we will briefly describe FORS (flexible QiganizationsJ, an environment for implementing computational paths that have been designed with the aid of TAO graphs. Specifically, FORS provides facilities to aid in (1) integrating people, tool-objects and data-objects into computational paths; and (2) building control organizations on top of the computational paths to control the flows of information along them.

The integration problem is difficult because the data-and tool -objects that one would like to connect often contain programs from different vendors, are written in different languages and styles, reside in different computers, have different interfaces and use different data formats. Integration environments, such as FORS, provide facilities or "smarts" to help overcome these difficulties. In most environments these smarts tend to be placed either with the data or with the tools but not both (see for instance (Daniell and Director, 1989)). We feel that some smarts fit more naturally with the data, others, more naturally with the tools. In particular, facilities for translation, browsing, detecting and correcting errors, and filling in missing bits of information, belong with the data; while information on how to use, repair and modify a tool, belongs rightfully with the tool. Therefore, FORS allows for its smarts to be divided between data- and tool-objects.

The control problem is difficult for two reasons. First, the control structures required for complex design problems are complex hierarchies (c.f. Section 2.8). However, the experience with automating such hierarchies is limited. In existing design systems automation covers at most the two lowest levels, as in the use of a software structure called a blackboard (Proceedings of the AAAI and Boeing Workshop on Blackboard Systems, 1987). All other control is manual.

Second, control structures need to be dynamically adjusted to account for unforseen contingencies. Invariably, programs will not work as expected, pieces of data will turn up missing, errors will be made, and unforseen opportunities will appear.

FORS has been designed to allow for the building of complex, dynamically adjustable control structures. However, as yet we have not taken advantage of this capability and its benefits remain to be evaluated.

The following material describes FORS' features in slightly greater detail.

Data-Objects

FORS allows for an expandable library of data-objects. Each of these objects can store one or more aspects and has the facilities to make the aspects available to operators. These facilities include translators (to make aspects available in representation schemes of the operator's choice), error correctors, editors, browsers and default generators (to fill in missing information when necessary). An example is a data-object that is under development for the circuits of electric power networks. When completed, this object will make its contents available in both diagramatic and tabular forms. Simple errors will be identified and rectified where possible. Typical values of missing information will be supplied when asked for. Both browsing and editing functions will be supported.

To create a **new** class of data-objects one must develop the mechanisms to support it—representation schemes, translators, etc. This can take a considerable amount of effort. Once the class has been established, however, individual members inherit the support mechanisms and are relatively easy to create.

Tool-Objects

FORS allows for an expandable library of tool-objects. Each of these objects contains a tool or programmed operator that may be written in a number of languages (currently, the list includes: Common Lisp, Fortran-77, C and OPS 5).

Each tool-object also contains a simple template whose contents describe the principal characteristics of the tool, which computer it resides in, how to use it, and which formats it prefers for its input and output data. If the usage patterns and data formats of a new tool are supported by the existing classes of data-objects, then adding the tool to FORS is as simple as filling out the template-a matter of a few minutes. Otherwise, modifications to the clases of data objects are required, and as has already been mentioned, these modifications can require a good deal of effort.

The Interface

FORS has a multi-window interface (Papanikolopoulos, 1989) that has been built on top of DPSK (Cordozo, 1989), a kernel for distributed problem solving, and on a graph display package (Vidovic, 1989). The interface represents each data-object and tool-object by an icon. By arranging these icons to form chains (Fig. 5), the user can create and execute arbitrary computational paths. All the underlying details involved in dealing with a distributed set of heterogenous computers and programs written in different languages are transparent to the user (provided that the computers are networked and use UNIX).

Automatic Planning and Execution Control

The interface can, ofcourse, be used by people but it can also be used by supervisory programs to construct and modify computational paths. At present, we have only one such program and it is rather simple-given starting and ending data-objects, it identifies all paths that link these objects through available tools. The distributed problem solving facilities that underlie the interface can accomodate much more complicated control structures, including trees and lattices of supervisory programs. The advantages that will accrue from the use of such structures is just beginning to be investigated.

Status and Plans

The basic design, interface and underlying distributed problem solving structure of FORS have been completed. We are proceeding to stock its libraries with objects that will allow the rapid construction of computational paths through a variety of disciplines. Currently, FORS contains about 15 data-objects and 17 tool-objects from the domains of civil, mechanical and electrical engineering. Within a year we expect to add twenty to fifty new objects. These additions will consist primarily of critics and conflict resolvers for the IBDE project and the other two concurrent design platforms under development, namely, the real time control of power networks and the design of automobile parts.

CONCLUSIONS

This paper has:

1. defined a set of terms, including aspects, operators, consistency and conflicts, with which to formulate the computational problems of concurrent design;
2. argued that, good concurrent design is equivalent to constructing integrated, conflict free, computational paths;
3. suggested that available and desired computational paths be visualized with the aid of TAO graphs whose nodes represent aspect-spaces and whose arcs represent design operators;
4. described FORS, an integration environment that is being developed for the rapid implementation of integrated, conflict free, computational paths.

As such, the paper provides the beginnings of a framework for handling problems in concurrent design.

REFERENCES

S. N. Talukdar, S. S. Pyo and Ravi Mehrotra, "Distributed Processors for Numerically Intense Problems", Final Report for EPRI Project RP 1764-3, March, 1983.

Petter Stoa, Sarosh Talukdar, Richard Christie, Lily Hou and Nikolaos Papanikolopoulos, "Environments for Security Assessment and Enhancement," presented at the Second Symposium on Expert Systems Applications to Power Systems (ESAPS'89), Seattle, WA, July 17-20, 1989.

Mark Sapossnek, Sarosh Talukdar, Alberto Elfes, Sergio Sedas, Moshe Eisenberger, and Uly Hou, "Design Critics in the Computer-aided Simultaneous Engineering (Case) Project," Technical Report, EDRC, Carnegie Mellon University, 1989, to be presented at ASME Winter Annual Meeting, San Francisco, CA., Dec. 11-14, '89.

James Daniell and Steven W. Director, "An Object Oriented Approach to CAD Tool Control Within a Design Framework," Technical Report, EDRC, Carnegie Mellon University, Nov. 18, 1989.

Proceedings of the AAAI and Boeing Workshop on Blackboard Systems: *Implementation Issues,* Seattle, WA, July 1987.

N. Papanikolopolous, TORS: Flexible Organizations," MS Thesis, Carnegie Mellon University, 1989.

E. Cardozo, "DPSK: A Kernel for Distributed Problem Solving," Ph.D. Thesis, Carnegie Mellon University, 1987.

N. Vidovic, D. Siewiorek, and F. Newberry, "A Graph Based Environment," Technical Report CMUCAD-87, 1987.

REFERENCES

S. N. Talukdar, S. S. Pyo and Ravi Mehrotra, "Distributed Processors for Numerically Intense Problems", Final Report for EPRI Project RP 1764-3, March, 1983.

Petter Stoa, Sarosh Talukdar, Richard Christie, Lily Hou and Nikolaos Papanikolopoulos, "Environments for Security Assessment and Enhancement," presented at the Second Symposium on Expert Systems Applications to Power Systems (ESAPS'89), Seattle, WA, July 17-20, 1989.

Mark Sapossnek, Sarosh Talukdar, Alberto Elfes, Sergio Sedas, Moshe Eisenberger, and Lily Hou, "Design Critics in the Computer-aided Simultaneous Engineering (Case) Project," Technical Report, EDRC, Carnegie Mellon University, 1989, to be presented at ASME Winter Annual Meeting, San Francisco, CA., Dec. 11-14, '89.

James Daniell and Steven W. Director, "An Object Oriented Approach to CAD Tool Control Within a Design Framework," Technical Report, EDRC, Carnegie Mellon University, Nov. 18, 1989.

Proceedings of the AAAI and Boeing Workshop on Blackboard Systems: *Implementation Issues,* Seattle, WA, July 1987.

N. Papanikolopolous, TORS: Flexible Organizations," MS Thesis, Carnegie Mellon University, 1989.

E. Cardozo, "DPSK: A Kernel for Distributed Problem Solving," Ph.D. Thesis, Carnegie Mellon University, 1987.

N. Vidovic, D. Siewiorek, and F. Newberry, "A Graph Based Environment," Technical Report CMUCAD-87, 1987.

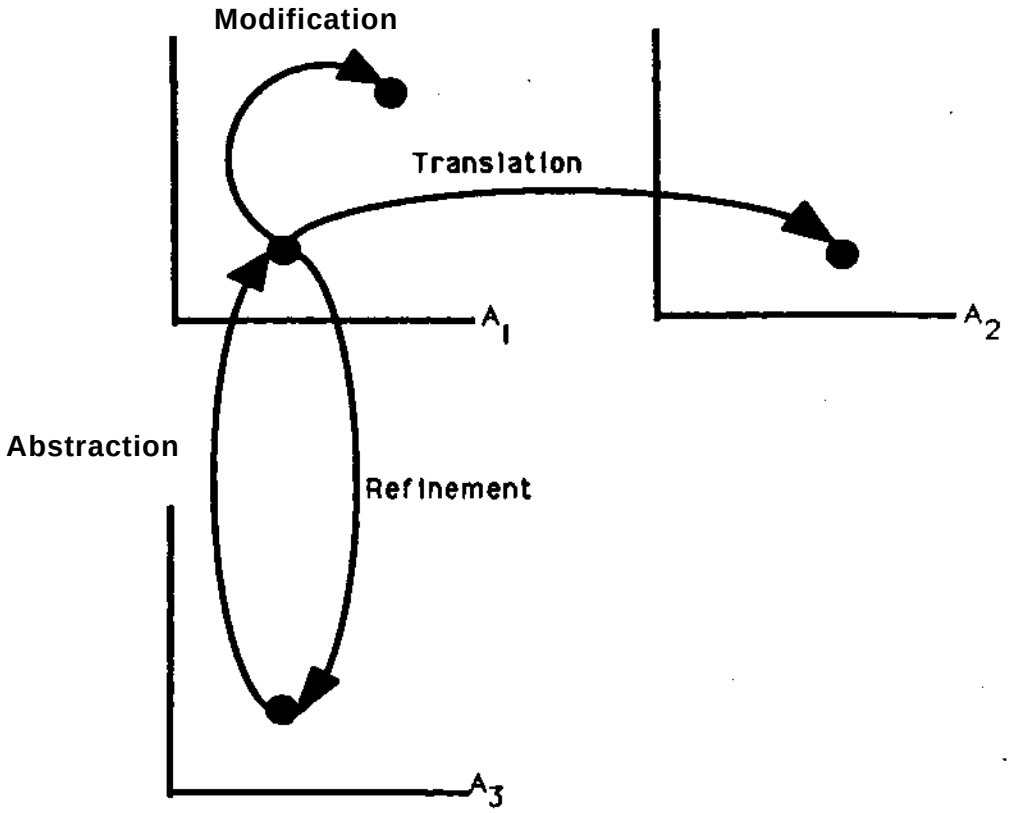

Fig 4: The mappings produced by four types of operators. Al and A2 are Informatlonally equivalent spaces. A3 has more Information contact than Al