Emergent Coherent Behavior of complex configurations through automated maintenance of dominance relations

N. John Habraken

Prof. Emeritus, Architecture, MIT, john@habraken.com

J. Willem R. Langelaan

Langelaan Architect, Mississauga, ON, Canada, willem@langelaan.com

ABSTRACT: When we change a configuration by displacing or removing one object in it, desired adjustment of related objects usually follows established relational conventions. If, for instance, we displace a wall in which a window is found, we will expect the window to remain in the wall. At the same time we expect to be free to shift the position of the window in the wall without adjustment of the wall. This a-symmetrical relationship is one of Dominance. Relations of dominance are the main reason for structural and functional coherence. CAD programs may have ad-hoc performance of dominance, like with the window in the wall, but do not know the general principle, nor do they allow us to inform the computer about dominance relations we want it to maintain.

Sub-ordinate behavior drives the system. Dominant behavior results by implication. The methods defining sub-ordinate behavior are encapsulated in the object class. The designer determines the unique behavior of an object instance by specification of attribute values. Data input per instance related to class allows the computer to perform adjustments in an instance-to-instance chain, maintaining coherence of a complex configuration without need of global checking of relational data.

This paper demonstrates a notation and interface system by which dynamic relations, including those of dominance, between any two objects can be conveyed to the computer, enabling it, when properly programmed, to follow up adjusting when we design.

The paper further elaborates the concept of dominance leading to computer supported autonomous editing of a virtual building model as intended by the proposed interface and resulting in emergent coherent behavior of its constituent elements.

KEYWORDS: Design, Emergent behavior, Dominance relations, bi-directional editing

1. INTRODUCTION

In object oriented CAD application software objects belong to classes. Classes are parametric with attributes and attribute values. A specific object is an instantiation of a class. It has specific values given to the attributes (variables) of the class and may also include specific functions of the class. Classes may be simple primitives such as integers, real numbers or characters, more complex things such as vectors and text strings or may be very complex (high level) assemblies of primitives or even other classes. A high level architectural class may represent a door in a frame complete with hardware. The location of the object is determined by the coordinates of its local origin in relation to the origin of the database.

Dominance has been explained as a pervasive relationship among objects in complex configurations like the built environment. [Habraken 1998]. (See also 3.3 in this paper.) It is partly based on physical constraints like gravity and containment, but also to a large extent by convention of use and interpretation. The concept gives rise to algorithmic principles by which dominance can be ‘understood’ by the computer.

In general use in practice precedes programming; The interfaced proposed here illustrates the use of dominance relations and may also clarify the concept and stimulate its being incorporated in CAD programs. There are always many possible interfaces to deal with a certain design aspect. In this case we have tried to conceive of a simple formula to be used in conjunction with visual object representation on the screen.

The interface is bi-directional in that changes in a relationship between objects show in the formula bar, while changes in the latter activate changes in the visual representation. Where terminology is needed we have tried to keep it as close as possible to the practitioner's experience, avoiding unnecessary mathematical vocabulary.

2. AN INTERFACE CONVENTION

2.1 Relations between objects

Given two objects in space, as shown in an on-screen design, a Distance Vector can be drawn from one object to the other. If drawn from A to B, it tells us about a relation of A relative to B. Many different Vectors can be drawn from A to B each giving another relational aspect. The length of a vector will be calculated by the computer and may be seen next to it on the screen:

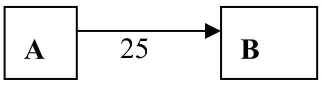

Figure 1, Relation of A towards B.

Once a vector is drawn, a formula will appear in a formula bar:

A, B = 25 1)

If we change the Vector value to say, 20, object A will translate along the Vector towards B over a distance of 5. The Vector value can be changed in the formula as well as in the design. If we displace one of the objects, the vector and the formula will adjust. If we draw another Vector between A and B or between two other objects, the first vector will disappear.

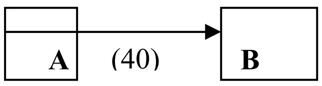

Figure 2. Negative vector value.

By convention, Distance vectors running outward from an object have positive values, those drawn inward across the object will be negative.

2.2 Sides of objects

To identify sides of an object in the formula bar, a convention of names for three pairs of opposite sides are proposed: Head and Tail, Left and Right, Spine and Belly, or, respectively H, T; L, R; S, B.

For easy reading, a black dot will be shown at the head of an object when a Vector is drawn from or

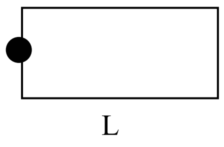

Figure 3. Identification of sides of an object.

towards it. When the head is given, only one side of one other pair needs to be identified for all six sides to be known. For instance, when side L in figure 3 is identified as well as the head, all other sides are identified. We will be free to establish the sides of objects as we prefer, or to follow generally accepted conventions.

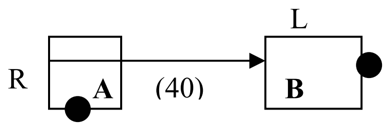

Figure 4. Relations by means of sides.

Figure 4, for instance, specifies further figure 2, and will show in the formula bar as:

Ar, Bt = (40) 2)

Here the sides are written in lower case because they specify the relation AB. Instead of drawing a Distance Vector by hand, it can be efficient to let the computer choose a default Vector based on an orthogonal system, to be corrected later. Clicking on A first and B next would produce the Vector of figure 1. But now the formula gives us the sides as well:

Al, Bt, = 25 3)

We could next alter Al into Ar, and obtain the Vector of Figure 4 with the (negative) length of (40).

2.3 Behavior profile (1)

In formula 3) we may replace the value of 25 with 'x', and we would get:

Al, Bt, = x? 4)

The question mark asks for a specification of the constraints on x. Double clicking on it we get a window in which we may write, for instance:

Al, Bt, = x > 10 5)

Saying in fact that the distance between the left side of A and the right side of B in figure 4 must be more than 10. This amounts to a statement of the behavior of A relative to B. The window contains the behavior profile of A. Relations of A to other objects may be entered there as well.

2.4 Internal relations

Once the sides of an object are identified, we can transform the object by moving opposite sides relative to one another. In figure 3, for instance, we can click Head and Tail, obtaining the corresponding default Vector:

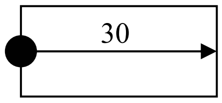

Figure 5. Relation of sides of an object

While the formula bar shows:

H, T, = 30 6)

This time the sides are written in capital letters because they are the objects in play. If we change the Vector value of 40 into 20, for instance, we will see the object shrink with H moving towards T because we have clicked H first. If we change formula 6) and write L instead of H, the computer knows relation LT makes no sense and will give us:

L, R, = 12 7)

This is a general way to find out dimensions of objects and change them.

2.4 Dominance relation

If we want to establish a dominance relation in which A is to follow B, we must click A first and B later to obtain the default Vector of Figure 1, accompanied by formula 1). We next underline the formula obtaining:

<u>Al, Br, = 25</u> 8)

Meaning that we want A to maintain that distance when B is displaced or when the side to where the Vector points is displaced. If next we displace object A the Vector value will change (or the other way around.) But if we displace object B, we will see object A move as well to maintain its position relative to B. This is typical dominance behavior.

Suppose later in the design we return to objects A and B, but now we click on B first, we will see that nevertheless the Vector will point from A towards B. Because the computer remembers B dominates A, and we cannot move it relative to A. Similarly, when we have a window and the wall in which it resides in a dominance relation, we will only get vectors from the window towards sides of the wall. See Figure 6.

Once we underline a formula the computer will maintain a dominance relation between the two objects for any Distance Vector that may be drawn or given constraints, maintaining the full behavior of the subordinate object relative to the dominant object. Removing the underlining in any statement will render sub-ordinate behavior inactive.

2.5 Behavior profile (2)

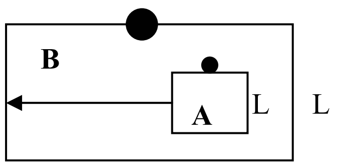

Figure 6. Window in wall

Suppose we now want to make sure the window stays within the wall. This requires a general statement about all possible relations between A and B. To make this, we click on A and B while keeping the shift key down, and get:

<u>A, B, = BP</u> ? 9)

In which BP stands for 'behavior profile' and the question mark indicates none is specified so far. To specify, we double click on BP and get a separate window in which A's BP is to appear. We may type there, for instance:

Al, Bl, = x	and	x > 0
Ar, Br, = x	and	x > 0
Ah, Bh, = x	and	x > 0
At, Bt, = x	and	x > 0

We may summarize this statement with a heading: for instance 'inside'. Next time when we ask for the general AB relation we will get:

<u>A, B, = BP 'inside'</u> 10)

Without a summary title, only BP will appear, and by double clicking on it, the behavior profile window will open. With a summary title we can quickly see what is going on and, in a second case of containment, the designer can simply write 'inside' in the general formula to make two new objects relate in the same way. Readymade behavior profiles under summary titles may be available for the designer, who remains free to edit them. For instance: by changing the zero value in any of the four relations, the window may stay away a certain distance from the wall's edge.

It may be that an object A is in relation with more than one other object. For each relation a BP can be written. When in such a case, holding down the shift key, we click on object A only, we get:

A, = BP 11)

And by double clicking on BP we will get the behavior profile window of A, giving us the specifications and summary names of all relations A maintains in our design.

2.6 Relations between classes of objects

The database held by a CAD program yields lists of all kinds of object classes available in the object library. Referring to those, the designer may call for relations between classes:

Class A, class B, = BP? 12)

And may enter sub-ordinate behavior of A by underlining the formula, or specify the BP for class A. In this way relations may be settled for all high back chairs relative to all work tables and dining tables. Or for certain kinds of windows in certain kinds of walls. Instead of these functional relations we may distinguish classes according to levels of intervention [Habraken 1998] like, for instance, partitioning following a base building and dominating furniture, or buildings following the site, or a curtain wall following a steel frame.

A single instance of an object may belong to several classes and thus inherit a fully specified behavioral profile. Having access to the individual object's BP, the designer can at all times edit it.

3. ALGORITHMIC ASPECTS AND BI-DIRECTIONAL EDITING

Bi-directional editing implies freely editing of a CAD database both deductively and inductively, and freely analyzing and evaluating a CAD database. In the current art, CAD application software can analyze a database on the basis of user specified criteria, such as square footage, building cost, etc. To evaluate a database on the basis of criteria and values is a manual and iterative editing process. Our proposal implies a structured approach towards automating such evaluation processes.

3.1 Objects

3.1.1 Object Geometry

The geometry of an object is specified by its morphology and topology. The morphological attributes qualify the shape. For example a curve. The topological attributes quantify the shape. For example the radius of the curve.

3.1.2 Object Space

An object has concrete space, which is the space filled by the object as a result of its geometry. It also has abstract space, which is the space required for it to function and the space required for the human user. For example the object geometry of a door is given by its height, width, and depth. Its functional space is the space needed for the sweep of its door leaf. Its user space may be additional space on either side of the door to allow access.

The total space that can be allocated to an object is the union of its concrete and abstract space.Functional space and user space alone can also be parametric classes. For example a corridor can be a functional space element. The associated user space may be smaller and is embedded inside the functional space.

3.1.3 Object Intersections.

In a design it may be found that the spatial relation between objects includes both their concrete and abstract spaces.

The Distance vectors in our proposal allow the maintenance of abstract space and concrete space of an object. For instance by setting a minimum distance from the object's concrete space to another concrete space or to an abstract space.

Relations will require editing. To automate such an evaluation requires that the geometric attributes of the abstract space are included with the parameters of each object. The nature of spatial relations may be an intersection of concrete space and another concrete space, concrete space intersecting with abstract space, and abstract space intersecting with abstract space.

3.2 Autonomous Object behavior

Behavior can be described as a deterministic system with in-put, through-put, and out-put. In a digital system these 3 phases would be: data transfer, data analysis, and internal data transfer to an encapsulated method. In the behavior sub-system the function of the data analysis phase is to determine to which encapsulated method the data should be passed. The encapsulated method may be an algorithm that edits the object geometry and/or edits the 6 parameters associated with the co-ordinates and orientation of the local origin within the Cartesian coordinate system of the model's database and/or outputs data to a selection of objects. The data analysis phase makes the object autonomous and it seems to have a will of its own. On the other hand, if the data analysis phase is not included with the sub-system, received data is executed as instructed and object behavior is pre-determined.

3.2.1 Reactive agents

Attribute values about internal object space and global position can be transmitted as low-level data between objects. In that case an object behaves as a reactive agent (Nwana 1996) that gathers data: "pulling data," and transmits data to a selection of other objects: "pushing data". Reactive agents do not possess internal, symbolic models of their environment. They act and respond in a stimulus–response manner to the present state of the environment in which they are embedded. Objects react in basic ways to dominance and behave as reactive agents. Therefore dominant behavior of objects can be coordinated and implemented with relatively simple stimulus–response class algorithms. Complex patterns of behavior emerge from these interactions between objects when observed as group.

3.3 Dominance behavior

Relative behavior of objects can be symmetrical or a-symmetrical. With symmetrical behavior a transformation of object A results in transformation of object B and vice versa. With a-symmetrical behavior a transformation of object B may not result in a transformation of object A, but may induce transformation of object C. The concept of dominance as illustrated in this paper captures this a-symmetrical behavior. A-symmetrical behavior was implemented in 1999 while developing new prototypes of high-level objects for a library of parametric objects [Langelaan 1999] for ArchiCAD [Graphisoft]. For example, a change in the section width of a doorframe results in a larger total width and height of the door object, but a change in the total width of the door object does not affect the width of the frame but implements a change of the width of the door leaf.

Dominance relations can be identified in different contexts. Although not always obvious, they are mostly conventional and therefore can be generalized with editable Behavior Profiles.

3.3.1 Design Level Sub-systems

These are generally governed by scale and construction sequence. [Habraken 1998]. Customary design levels are those for Urban Design, Site Design, Building Design and Room Design. More recently, a Fit-out design level has emerged in commercial and residential building [SAR 1978]. Dominance between levels is fairly obvious from design experience.

3.3.2 Group Aspect System

Within each design level, objects with a common aspect can be related as logical groups, such as mechanical, structural, electrical, egress, furnishing, storey, etc. Dominance is not obvious.

3.3.3 Class Aspect System.

This aspect system coordinates objects belonging to different classes. Class dominance is not obvious.

3.3.4 Custom Relationships

Design intention may lend sets of objects a relationship with custom dominance. For instance, object A always remains at the centre line of object B. Special geometric relationships could be specified with abstract relationship object classes.

3.4 Possible Applications

Capacity to deal in a systemic way with dominance relations allows new ways of computer supported editing and design development:

3.4.1 Fuzzy Choices

When the output of a logical process is discrete the process can be automated with Boolean algorithms. If a process encounters an array of possibilities whose choice is not stochastic neither objective nor decisive following iteration, but requires weighting and averaging then the logical process is fuzzy. At such occurrences an automated process must be interrupted and the question with its array of choices must be presented in an input window on the screen. For example questions about value sets of attributes, proportional relationships, dominance, etc. It must be noted that fuzzy logic reasoning algorithms may be possible to solve some of these problems.

3.4.2 Intelligent Editing

Intelligent Editing requires methods that maintain a geometric or positional relationship between discreet objects. The relationship can be inherited from the parent class or custom specified. The value of the relationship can be discrete "A on the centerline of B" or may be fuzzy "A near B" where 'near' has the value set, near={min(n),max(m)}. For example, when the geometry or position of a wall object is edited, the related doors and windows remain embedded in the wall. Or if a kitchen cabinet is placed on a floor plan it will automatically position itself against the nearest wall.

3.4.3 Bi-directional Editing

Bi-directional editing of an object occurs when attributes that have a reciprocal relationship can be freely edited, and the values of the related attributes are automatically updated and implemented.

Moreover, in sets of objects that have a reciprocal relationship, any object can be freely edited and attribute values of related objects are automatically updated and implemented.

Finally, criteria can be selected for analyzing the model database; the result can be edited and attribute values of related objects in the database are automatically updated and implemented.

4. CONCLUSIONS

The proposed instrument performs in a uniform way a number of functions already available like certain groupings, containment, shrinking or stretching and translation of objects. In addition it adds important new aspects having to do with relational behavior including dominance.

In design, basic relations are rather simple, but relational chains can become extremely complex. With the proposed instrument coherent control of this complexity may become possible, leading to greater efficiency and smoother coordination among parties in control of different sub-systems.

The instrument will render the computer relationally intelligent and capable of tracing strings of adjustment in case of local design change to check on possible conflicts between sub-systems; list object space violations; evaluate dominance of listed objects and generally monitor and help control dynamic relations among objects.

Because relational data are connected by an instance-to-instance chain, control of complex configurations can take place without global checking of relational data.

With the help of this instrument, we may learn more about behavior of complex systemic organizations by exercise and research. This, in turn, may enhance our theorizing on form behavior in general and building design in particular, and may lead to automated editing of a virtual building model.

5. REFERENCES

[Habraken 1998] N.J. Habraken, *The Structure of the Ordinary*, MIT Press, 1998

[Nwana 1996] H.S. Nwana, *Software Agents: An Overview*, Knowledge Engineering Review, Vol. 11, No 3, Cambridge University Press, 1996

[Langelaan 1999] J.W.R. Langelaan, *MasterLibrary 7.0*, unreleased prototype, Langelaan Architect, 1999

[Graphisoft] *ArchiCAD*, Graphisoft R&D Rt., Budapest, 1982-2003

[SAR 1978] Stichting Architecten Research, *Levels and Tools*, in: Open House International, vol.3, no.4