A NOTATION FOR THE VISUAL SPECIFICATION OF GEOMETRIC RELATIONS IN RULE-BASED USER INTERFACE DEVELOPMENT ENVIRONMENTS

Andrey K. Yeatts
Department of Computer Science
University of Arizona

Scott E. Hudson
Graphics Visualization and Usability Center
College of Computing
Georgia Institute of Technology

ABSTRACT

This paper describes a new visual notation for specifying geometric relationships (for example, in a user interface presentation application). This notation is designed to provide a centerpiece for the visual specification of predicates and rules in the BluePrint rule-based user interface development environment. The notation, while simple and using only a handful of operator symbols, is extremely powerful and expressive. It operates in an intuitive fashion using analogies to the physically based concepts of alignment and measurement to express a wide range of linear relationships between presentation objects, and can be used to express dynamic as well as static properties.

INTRODUCTION

In recent years, a number of tools have employed rules of various sorts to support user interface development. For example, the ITS system [Benn89, Wiec89, Wiec90] uses rules to automatically construct user interface presentations from the structure of application data. The UIDE system [Fole89] employs rules in the form of pre- and post-condition predicates in a number of different ways including the control of run-time appearance and behavior of an interface [Gies92]. The Object Lens and OVAL systems [Lai88] use rules to support tailorability of interfaces and applications by end-users. Finally, our own previous work [Huds91] has sought to integrate rule-based inference techniques into direct manipulation style user interface builders.

As indicated by the diversity of their uses, rules offer some significant advantages as a method for specifying computations. For example, individual rules are normally quite understandable, employing a simple if-then structure that indicates an action (such as the assertion of postconditions) and the circumstances under which it is to be executed or *fired*. In addition, rule sets are normally easily extensible. Typically new rules can be added to existing rules without special integration steps. As a result, rule-based systems are good at taking on evolutionary roles where increasingly specialized behavior is added over time.

However, many systems that have taken a rule-based approach have suffered from at least one drawback — the lack of an easily accessible visual notation for rules. These systems have often used programming language-like textual notations for rules or have simply hidden the rules from view (a notable exception is [Bell91]). This lack of an adequate visual notation for rules means that a number of these systems have been unable to exploit the advantages inherent in visually oriented tools such as interface builders (see for example [Card87, Myer89, Huds90]).

This problem is particularly acute for systems designed to work in geometric settings, for example, systems that apply inferencing techniques in the construction of user interface presentations [Meye86, Meye87, Meye89, Sing88, Sing89]. In these visual, or geometrically oriented domains, the objects, relationships, and actions that contribute to the construction of rules are often most easily expressed and manipulated in a visual manner.

The work described in this paper is designed to overcome this problem by providing a visual notation for expressing a wide range of geometric relationships between objects. This notation is specifically designed to support the specification of rules, allowing both pre-condition predicates and post-condition assertions or actions to be cleanly expressed.

NOTATIONAL PRINCIPLES

The design of the BluePrint visual notation is based on several notational principles including: explicit representation of all relationships, avoidance of strict WYSIWYG constructs, elimination of coincidental meanings, use of a minimal set of operators, and use of minimal hidden machinery. Each of these principles are described below.

Explicit representations

Each object and relationship expressed in the notation should have a visible manifestation. A number of previous systems (particularly those

This work was supported in part by the National Science Foundation under grant IRI-9015407.

taking *by-example* approaches [Meye87, Kurl91]), while directly representing the objects of interest, have provided no permanent representation for relationships that have been established between objects. While this approach may be adequate during initial specification, it is problematical if any modification or reuse of specifications is being supported.

This principle leads to the more direct corollary that a specification should be completely understandable from its printed representation and never rely solely on feedback or other dynamics to express meaning.

Avoidance of strict WYSIWYG constructs

WYSIWYG specifications have an intuitive appeal because of their directness. However, for geometric specifications that go beyond the simple size and placement of objects, and attempt to support expression of more abstract and semantically rich relationships, typical WYSIWYG notations actually begin to violate principles of direct manipulation. In particular, these notations typically provide no visual representation for the key objects of interest — the relationships between objects.

While it is possible to try to embed these representations in a WYSIWYG presentation (see for example [Huds91]), our experience shows that even with extremely compact representations, there is simply not enough space available in most situations to adequately express even moderately complex relationships. In addition, when dynamic as well as static aspects are being considered, the particular sizes and positions found in the notation are normally only instances of a larger class and take on lesser importance.

Rather than restrict the understandability of relationships to that of size and position, the notation described here avoids use of a strict WYSIWYG approach.

Elimination of coincidental meanings

As specifications increase in size and complexity it is important to avoid the occurrence of *coincidental meanings* — that is the assignment of meaning based on the coincidental placement of objects. As a corollary to the previous two principles this implies that geometric attributes (e.g., size and position of objects) should be given "don't care" values by default, rather than assuming specific values from their placement in the specification.

Use of a minimal set of operators

This principle is based on the need for simplicity. Notational style should remain consistent throughout the specification, with the fewest different operator symbols being employed. In general, if less specialized meaning is expressed by operator symbols, then more specialized meaning can be imparted by the geometric configurations themselves (which are inherently more direct).

Use of minimal hidden machinery

As a final principle, it is important to avoid hidden components and machinery. This implies among other things that more complex operations should typically be expressed in terms of simpler primitives.

BASIC NOTATION

In the physical world, we naturally employ notions of position, alignment, measurement, and comparison. In particular, we are accustomed to mechanically manipulating the position of real objects to make measurements and comparisons. For example, to cut two sections of pipe to the same length we would align their ends use one pipe as a guide for cutting the other. In the notation described here we attempt to exploit this familiar behavior.

The notation consists of a small set of operators to compare graphical attributes, and an operator that allows the introduction of an assumption (such as alignment). Each notational construct has both an intuitive, or analogous, interpretation and a more formal mathematical interpretation. For example, as will be discussed below, the *assumption* operator can be seen as applying a transformation (or *linear map*) to a set of comparison operators.

Objects

For the examples used in this paper, we will assume that objects can be characterized primarily by the four graphical attributes (x1, x2, y1, and y2) that make up their bounding boxes. As a result, objects will be represented by (or more generally simply enclosed in) a rectangle. Non-geometric attributes, such as highlighting, color, etc., will be expressed directly by modifying the attributes or example object contained within the bounding box, (but will not be discussed in detail in this paper).

According to the principles of the last section, the relative placement of objects implies no meaning in itself. For example, the apparent placement of object B to the right of object A does not specify or imply any actual relation of A to B.

The Mouse Object

To support the description of dynamic behavior based on user input, we introduce a special object, the *mouse object*, to model the X and Y attributes of

the pointing device and its button state. The notation corresponding to four mouse states is illustrated in Figure 1. The cross hair at the top of the mouse symbol is used to denote the X, Y position of the mouse, while the notation within the mouse is used to indicate a button press event, a button release event, and movement with the button in the up and down states, respectively.

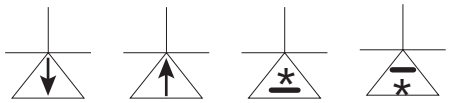


Figure 1. Mouse states press, release, and movement with down and up button values.

Ruler Lines

To refer to constants, indicate a common value, or allow drawings to be extended across more space for clarity, we use ruler lines. Figure 2 shows ruler lines denoting the value 100 and the symbolic value S.

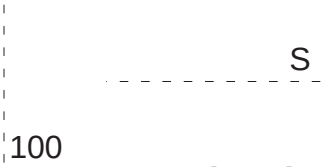


Figure 2. Two Ruler Values

Operator Symbols

The overall notation uses three basic operator symbols and a negation modifier as shown In Figure 3.

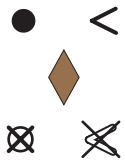
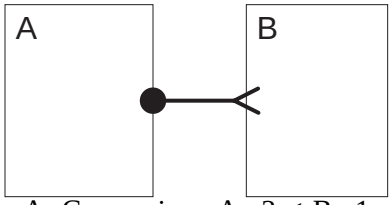


Figure 3. Operator Symbols

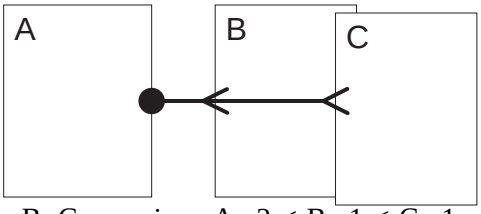
The top two operator symbols (left to right) are comparison operators expressing equality and less than respectively. The third operator is the assume operator described below. Finally, a negation may be applied to either of the first two operators by placing an X over it as shown at the bottom of Figure 3. To improve readability when negation is applied, we "hollow out" each comparison operator.

Comparison Operators

To describe a comparison of two or more attributes, we link the attributes with graphical symbols denoting the relative ordering of the attributes. In the example of Figure 4A below, we show the notation for expressing the relationship A.y2 < B.y1. An ordering of three or more attributes can be illustrated similarly, as in Figure 4B.



A. Comparison A.x2 < B.x1

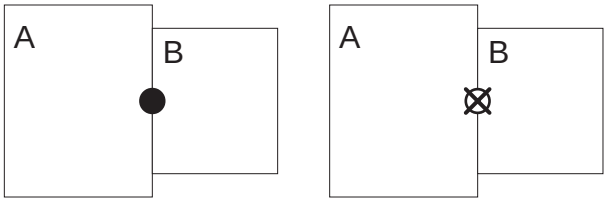


B. Comparison A.x2 < B.x1 < C.x1

Figure 4.

To describe the equality of the attributes of A.y2 and B.y1, we adjoin the attributes and place an equal symbol over the equal attributes (Figure 5A).

To negate a test, a cross is placed over the comparison operator, as in Figure 5B.



A. A.x2 = B.x1 B. A.x2 ≠ B.x1

Figure 5.

Figure 6 shows the use of a ruler line to extend a value for several common references.

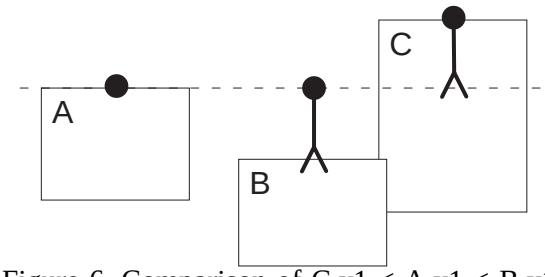


Figure 6. Comparison of C.y1 < A.y1 < B.y1

TRANSLATIONS AND DIFFERENTIAL MEASUREMENTS

When object widths are measured in the physical world, two objects are aligned to a common reference point and the opposite ends are compared for distance to the common point. We exploit this

model for making differential comparisons in our notation.

The Assume Operator

To define a common reference for the purposes of measurement, we use the assume operator. This operator acts by introducing an assumption of alignment (e.g., equality) as in "assuming that X and Y are the same...". The mathematical effect of the operator is to apply a mapping — that is, to rewrite one or more tests making them relative to the points assumed to be common. Figure 7 below compares the width of objects A and B. Specifically, the test A.x2 < B.x2 is made relative to the assumption that A.x1 = B.x1. Another way of viewing the diagram is that if two objects, A and B, happened to have their x1 coordinates coincident as indicated by the assumption, then the test would hold as shown.

Figure 7. Comparisons of widths of A and B using the assume operator

The Translation Semantics of Assume

More formally, an assume operator defines a map for each graphical object. Each map transforms points in one object to a common reference point by means of a translation. If the common reference point is a ruler line, then the translation is to that value, otherwise the value 0 is used. For example, in the Figure 7 above, the map $M_{A.x1}(x) = x - A.x1$ transforms points of object A into points relative to 0, and map $M_{B.x1}(x) = x - B.x1$ transforms points in object B into points relative to 0. The tests on the x2 attributes of A and B are then rewritten by applying those maps to the comparison $A.x2 < B.x2$, resulting in $M_A(A.x2) < M_B(B.x2)$, or, substituting the map definition $A.x2 - A.x1 < B.x2 - B.x1$, which is just $A.width < B.width$.

Combining the assume operator and rulers with values allows us to compare widths to specific numerical values. In Figure 8, the object A's height is compared to the value 80. Again, we might read the diagram as "if a matching object A were to have its y1 attribute equal to 0, then its y2 attribute would be less than 80."

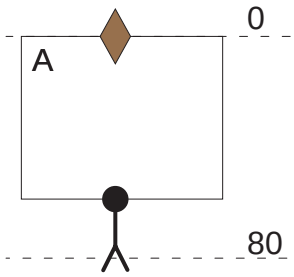


Figure 8. Comparison A.width < 80

Square Configuration

In another application of this technique, we can compare the object to itself to see if its width and height are the same. In Figure 9, an instance of the object A is rotated 90 degrees with respect to another copy of A, with its x1 and y1 edges made relatively equal to each other and the x2 and y2 edges compared for equality.

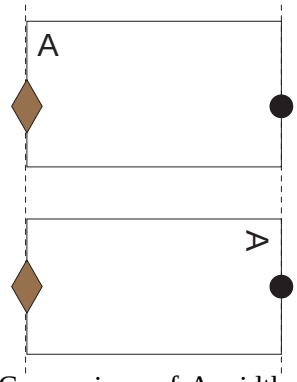


Figure 9. Comparison of A.width = A.height

Sum of Widths Comparison

By the use of these maps, several assumption points can be used in a configuration to "stack" objects for comparisons. In Figure 10, we describe a test for $A.width + B.width < C.width$. The progression through map application follows the figure.

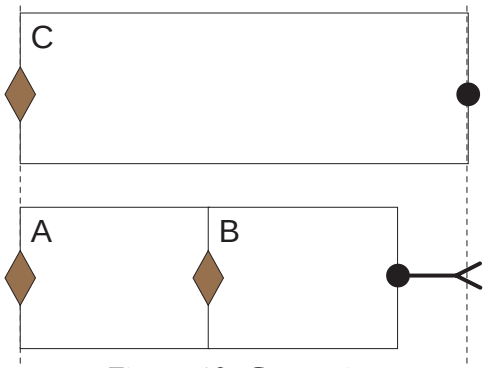


Figure 10. Comparison $A.width + B.width < C.width$

The initial test is $B.x2 < C.x2$ which is transformed by the maps $M_{B.x1}$ and $M_{C.x1}$ into

$$M_{B.x1}(B.x2) < M_{C.x1}(C.x2)$$

or:

$$B.x2 - B.x1 + A.x2 < C.x2 - C.x1.$$

The map $M_{A.x1}$ is applied to A.x2 to yield:

$$B.x2 - B.x1 + M_{A.x1}(A.x2) < C.x2 - C.x1,$$

or:

$$B.x2 - B.x1 + A.x2 - A.x1 < C.x2 - C.x1.$$

which, finally, is the same as :

$$B.width + A.width < C.width.$$

TRANSFORMATION SCOPE

As we have seen, if an assumption is applied to an object, the other attributes of the object are considered to be in the scope of the assumption. We may extend this scope to apply to other "non-member" attributes by completely enclosing the object to be mapped inside the object about which we are making assumptions.

In Figure 11, two maps are defined by the assumptions that $A.y1 = B.y1$. These assumptions induce transformations $M_{A.y1}$ and $M_{B.y1}$. By the enclosure scoping mechanism, the maps are applied to $C.y1 < D.y1$ to describe the condition $C.y1 - A.y1 < D.y1 - B.y1$. Thus, an object's graphical representation provides both a transformation and a scope for the transformation.

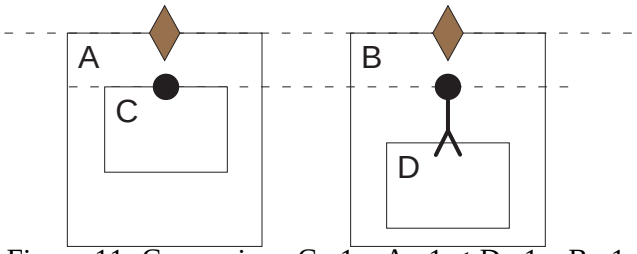


Figure 11. Comparison C.y1 - A.y1 < D.y1 - B.y1

To describe the desired scope of map application from objects that may not have a bounding rectangle (e.g., a ruler or the mouse), we can create explicitly a scope by the use of a "virtual object" that allows enclosure of the attributes to which the map is to be applied. In Figure 12, the map induced by the assumption that the mouse y value is equal to 10 is applied to the condition that the mouse x value is between 5 and 15.

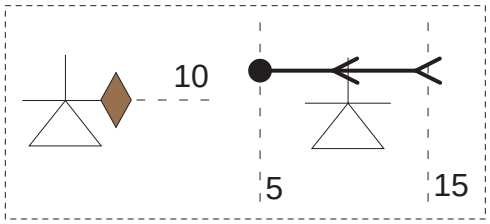


Figure 12.

The map $M_{mouse.y}(z) = z - mouse.y + 10$ applied to $5 < mouse.x < 15$ yielding:

$$5 < mouse.x - mouse.y + 10 < 15.$$

If two assumptions are made about the X and Y axes, then those assumptions are applied preferentially to their axes; see Figure 28 for an example.

Comparison to a Unit Slope Diagonal

One use of the enclosure scoping describes the configuration of a pair of attributes lying equidistant in x and y from another pair of x and y attributes. As in the previous mouse example, we may use more than one copy of an object in a configuration; in Figure 13, we condition B.x1 and B.y1 to lie equally distant from A.x1 and A.y1, describing the path along the line of slope -1 passing through (A.x1, A.y1).

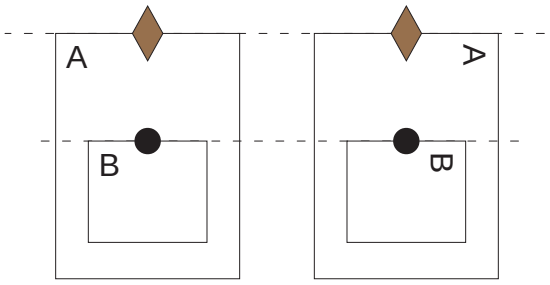


Figure 13. Comparison $B.y1 - A.y1 = B.x1 - A.x1$

Composition of Assumptions

To create more complicated relations, we may nest assumptions. This is accomplished by graphically nesting the assumptions, which are applied from the inside out. An assumption nested inside one assumption scope applies to other objects enclosed at the same nesting level. For example, nesting the assumption of Figure 12 inside Figure 13 (shown below in Figure 14) results in the test:

$$5 < mouse.x - A.x1 - (mouse.y - A.y1) + 10 < 15$$

which tests whether the mouse is within 5 of the unit diagonal passing through the point (A.x1, A.y1).

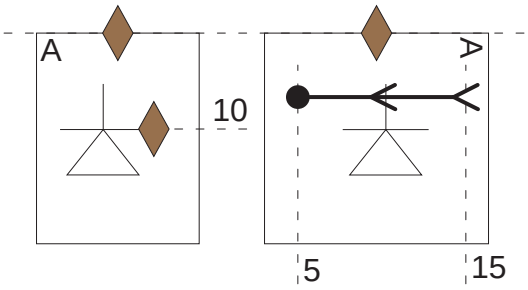


Figure 14. Composition of assumptions.

PROPORTIONALITY AND LINEAR COMPARISON

Carrying physical measurement models a step further, we describe proportionality as an extension of the assume operator discussed earlier. We may intuitively pose proportionality as comparing objects that have been stretched to occupy similar intervals. Thus, the span of two object attributes (or values along an axis) denotes a stretched interval that may be considered equal to another such interval.

Proportionality

Here, we use two of the assume operators to logically "pin" two attributes together, thus creating maps from one interval to another. By analogy, we are making the assumption that "were these two intervals identical, the following would hold." In Figure 15 below, we make the assumption that A.y1 and B.y1 are the same, and that A.y2 and B.y2 are the same. The map M_A takes points in the interval [A.y1, A.y2] and maps them to points in a reference interval [0,1] via the transformation $M_A(y) = \frac{y - A.y1}{A.y2 - A.y1}$. Likewise, the map M_B takes points in the interval [B.y1, B.y2] to points in the same reference interval. Note the use of ruler lines to represent the relative points of reference.

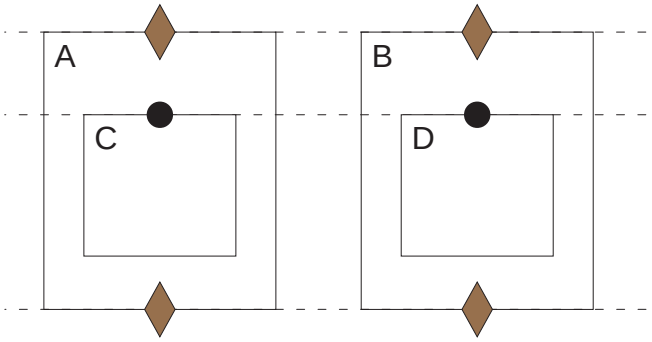


Figure 15. Comparison of C.x1 = D.x1, proportionally relative.

As in the nested transformations case, we signify the application of M_A and M_B to points by enclosure of the attributes to be mapped inside the objects A and B themselves. In Figure 15 below, C.y1 is tested against D.y1 proportionally to the y span of objects A and B. After the applications of the maps, the test $C.y1 = D.y1$ is transformed to

$$M_A(C.y1) = M_B(D.y1) \quad \text{or} \quad \frac{C.y1 - A.y1}{A.y2 - A.y1} = \frac{D.y1 - B.y1}{B.y2 - B.y1}.$$

Scroll Window Example

An obvious application of this device is the scrolling area of a window controlled by a scroll bar thumb or pointer. There is a direct modeling analogy between the file span and the span of the scroll bar, and the position of the thumb and the file window. In Figure 16, this situation is described as "assuming that the file and scroll bar occupied the same reference interval," the file window and scroller thumb would be at the same point.

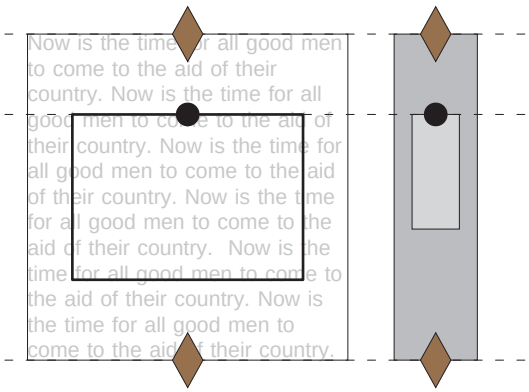


Figure 16. Scroll bar with thumb and window tops proportionally positioned.

Another common configuration for scroll controllers is that the scroll thumb top and bottom are proportional to the window top and bottom. This is described in Figure 17 below.

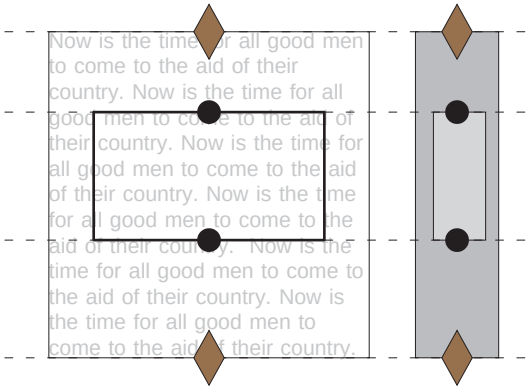


Figure 17. Scrollbar with proportionally sized thumb and window.

Comparison to Diagonal or Arbitrary Line

If we take the comparison to unit diagonal diagram (Figure 13) and make the distances proportionally equal, rather than strictly equal, we may describe a point lying on an arbitrary diagonal. The diagram of Figure 18 below is almost identical to that of Figure 13, but we make the distances proportional by using interval maps, rather than translation maps.

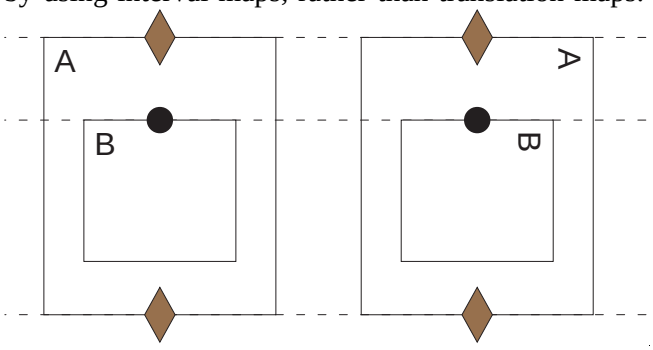


Figure 18. Comparison of point (B.x1, B.y1) to line through (A.x1, A.y1),(A.x2, A.y2)

Comparison to Half Plane

A slight modification to Figure 18, so that inequality, rather than equality of points is tested, results in requiring the X distance offset to be proportionally greater (with respect to A) than the Y distance offset. This is illustrated in Figure 19.

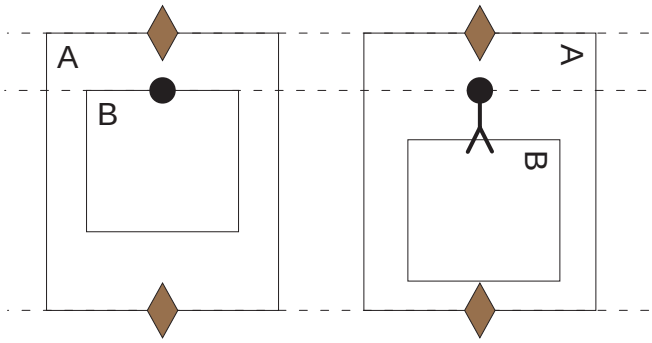


Figure 19. Configuration of (B.x1, B.y1) lying in upper right hand half plane

Though a full exposition is beyond the scope of this paper, proportionality is an extremely versatile device, allowing the visual expression of such concepts as aspect ratio and area.

USING THE NOTATION IN AN INTERFACE BUILDER

The preceding sections have considered how the notation can be used to create a wide variety of descriptions of geometric relationships. This section will consider how these descriptions can be employed in a rule-based user interface tool, specifically the BluePrint system under construction at the of University of Arizona. BluePrint describes both static and dynamic aspects of a user interface in terms of if-then rules. These rules describe transitions between object states and object configurations. As objects in the BluePrint system receive updates for their attributes this may trigger the application of rules that can create and delete components, modify the state of existing components, and transform object configurations. Rules that can be expressed in geometric terms are specified using the visual notation described here.

In general, this notation can be used in several different ways in a rule-based system. Most importantly, it can be used to provide *preconditions* or *predicates* that describe the test portion of a rule (the conditions under which rules fire) and it can be used to indicate the action portion of a rule with a form of *postcondition assertion.*

Both test and action portions of a rule in BluePrint contain visual specifications describing an object layout using the notation above. When an object configuration described in the test portion of a rule is found to hold true for some part of the interface, the action that follows the test is executed. Multiple configurations or descriptions in a rule head form conjunctions ("ands"). To specify disjunctive conditions ("ors"), multiple rules are used.

A Graphical Editor Example

To illustrate the character of a BluePrint specification, we present a complete specification for a button that creates a new line object and the line's subsequent manipulation by the mouse, such as might be found in a graphical object editor.

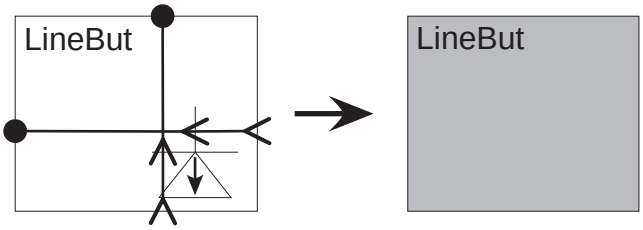


Figure 20. Rule enabling line button

The test (left hand side) of Figure 20 requires a mouse down event to occur within the bounds of a line button object. The action taken is to highlight the line button. The next two rules (Figures 21 and 22) return an active button to an inactive state when the mouse is outside it. Note the use of two rules to implement disjunction.

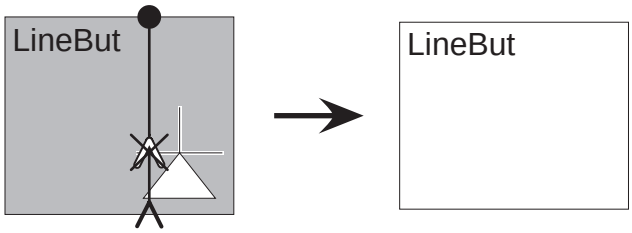


Figure 21. Rule disabling button when mouse is outside button.

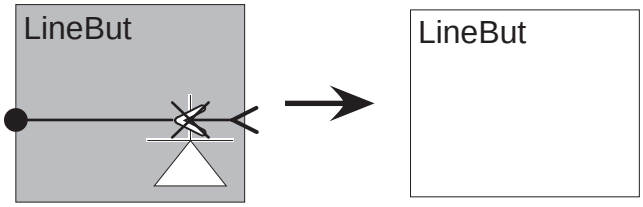


Figure 22. Rule similar to 21 above.

The button's action (creating a line) is performed when a mouse up event is received within an active button. The button is made inactive (by un-highlighting) and a line object is created at position (10, 10) as illustrated in Figure 23.

The remaining rules of the interaction specify how the mouse object can manipulate the line object. The interactions allow changing the line's (x1, y1) position. Figures 24-26 illustrate rules for selecting then moving the line as a whole, while Figures 27-29 show the rules for stretching the line.

The selection rules, shown in Figures 24 and 27, look for mouse down events in different configurations and change the appearance of the line to indicate its activated state. The rules following the selection rules update the line attributes to implement an appropriate drag operation, and the final pair of rules await mouse up events to de-select the line and end the interaction.

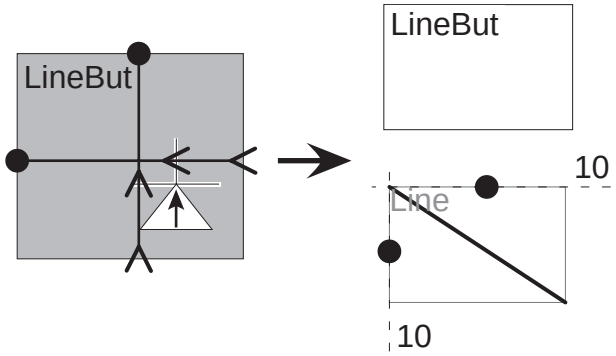


Figure 23. When the mouse button is released, the line button is un-hilighted and a line object is created.

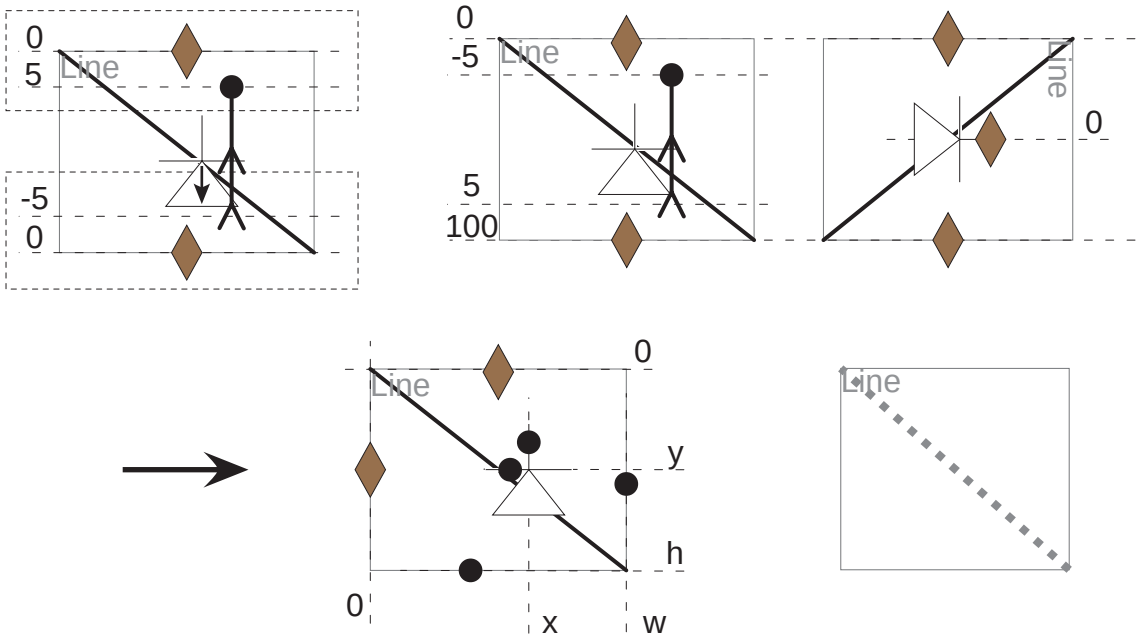


Figure 24. Specification initiating line's movement without changing its length or width.

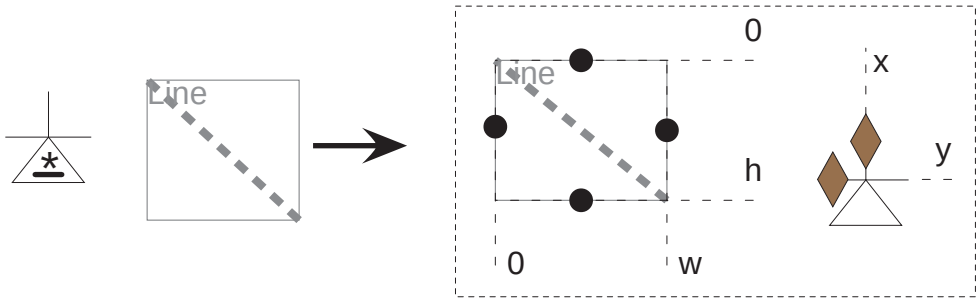


Figure 25. On mouse movement in down state with an activated line, adjust the line accordingly.

In the rule of Figure 24, the first condition specifies that a mouse down event occurs in within the central portion of the line (e.g., not within 5 units of the y component of the ends of the line). The second condition insures that the mouse is within a proportional distance of 5% from the line.

The mathematical interpretation of the first condition is: $line.y1 + 5 < mouse.y < line.y2 - 5$. Scoping objects allow the use of two separate translation assumptions — one for each end of the line.

To understand the mathematical interpretation of the second condition, we first examine the right hand part which introduces an assumption that $mouse.x = 0$, or a map: $M_{mouse.x}(x) = x - mouse.x$.

This map applied to the test $-5 < mouse.y < 5$ yields

$$-5 < mouse.y - mouse.x < 5.$$

The proportionality assumptions applied between the two components yields maps of:

$$M_{line.x}(x) = \frac{x - line.x1}{line.x2 - line.x1} \cdot 100 + 0$$

and

$$M_{line.y}(y) = \frac{y - line.y1}{line.y2 - line.y1} \cdot 100 + 0$$

that are then applied to the enclosed attributes, yielding the final test:

$$-.05 < \frac{mouse.y - line.y1}{line.y2 - line.y1} - \frac{mouse.x - line.x1}{line.x2 - line.x1} < .05$$

.

More intuitively, we read the notation as: "if a given line object happened to have the dimensions (0,0),(100,100), and the mouse x value were 0, then the mouse y value would lie in the interval [-5,5]."

On the right hand side of the rule, the actions taken by the rule highlight the chosen line object and establish offsets x, y, w and h to be used later by the position update rule.

The rule illustrated in Figure 25 updates the line's x1, x2, y1 and y2 coordinates to drag the line as a whole. Note the similarity to the action appearing in the rule of Figure 24. The specifications differ only in the assumptions applied, i.e., assuming that the mouse is at (x,y) as opposed to the previous assumption that the Line object was positioned at (0,0).

The final rule illustrated in Figure 26 returns the line to an inactive state on a mouse up event.

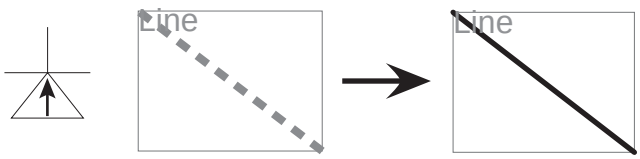


Figure 26. On mouse up with active line, inactivate line.

The rules of Figure 27 to Figure 29 describe a similar interaction that only modifies the x1 and y1 values of the line (i.e., stretches it). A different line appearance is used to distinguish the active state of the previous interaction (Figures 24-26) and that of Figures 27-29.

CONCLUSION

This paper has presented a very simple but expressive visual notation for describing geometric relationships and its application to rule specification in the BluePrint user interface development system. With this notation it is possible to express a rich set of geometric relationships and build rules governing visual behavior in a visual notation.

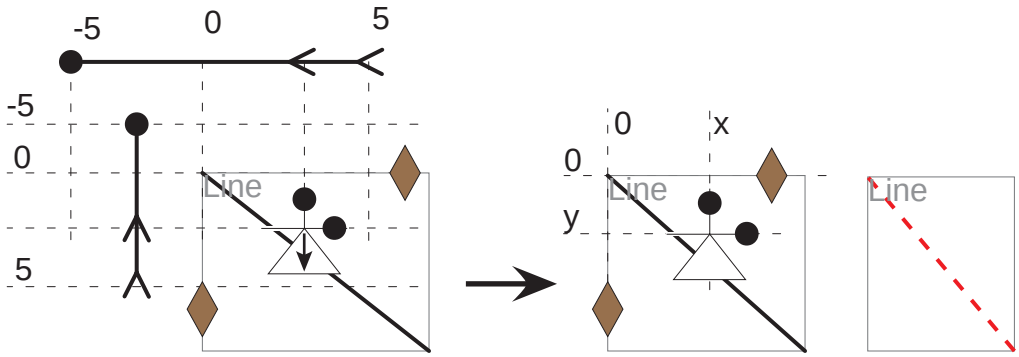


Figure 27. On down event within 5 units of upper left corner of line, activate line for (x1,y1) modification.

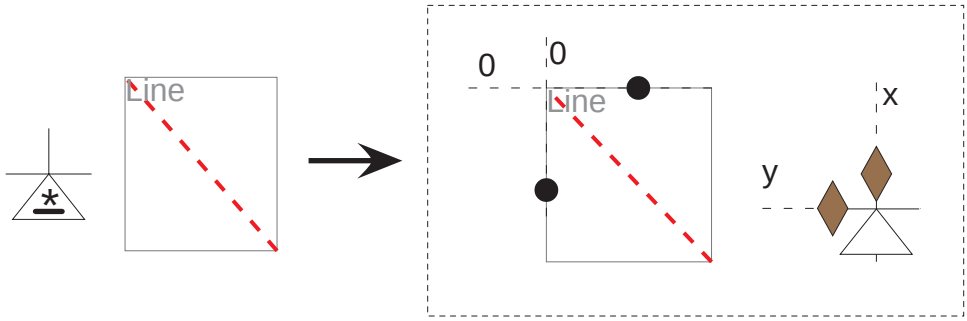


Figure 28. On mouse movement during button down, update line object's x1,y1 attributes.

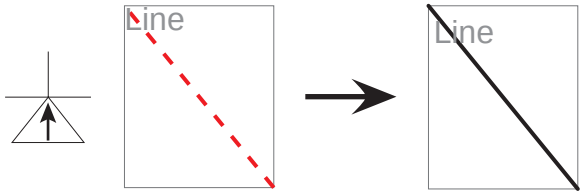


Figure 29. Rule returning line to the inactive state.

REFERENCES

[Bell91] B. Bell, "ChemTrains: A Visual Programming Language for Building Simulations", University of Colorado Technical Report CU-CS-529-91, June 1991.

[Benn89] W. Bennett, S. Boies, J. Gould, S. Greene, C. Wiecha, "Transformations on a Dialog Tree: Rule-Based Mapping of Content to Style", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, November 1989, pp. 67-75.

[Card87] L. Cardelli, Building User Interfaces by Direct Manipulation, *Digital Systems Research Center Tech. Report*, October, 1987.

[Fish92] G.L. Fisher, D.E. Busse, D.A. Wolber, "Adding Rule-Based Reasoning to a Demonstrational Interface Builder", *Proceedings of the ACM Symposium on User Interface Software and Technology*, November 1992, pp. 89-97.

[Fole88] Foley, J., Gibbs, C., Kim, W. Kovacevic, S., "A Knowledge-Based User Interface Management System", *Proceedings of CHI'88*, April 1988, pp. 67-72.

[Fole89] J.D. Foley, W. Kim, S. Kovacevic, K. Murray, "Defining Interfaces at a High Level of Abstraction", *IEEE Software*, vol 6, no 1, January 1989, pp. 25-32.

[Gies92] D.F. Gieskens, J.D. Foley, "Controlling User Interface Objects Through Pre- and Postconditions", *Proceedings of CHI '92*, May 1992, pp. 189-194

[Huds90] S.E. Hudson and S.P. Mohamed, Interactive Specification of Flexible User Interface Displays, *A C M Transactions on Information Systems*, Vol. 8, No. 3, July 1990, pp. 269-288.

[Huds91] S. Hudson, A. Yeatts, "Smoothly Integrating Rule-Based Techniques Into a Direct Manipulation Interface Builder", *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 145-153, November 1991.

[Kurl91] D. Kurlander and S. Feiner, Inferring Constraints from Multiple Snapshots, *Columbia University Technical Report*, May 1991. Tech. Rep. CUCS 008-91

[Lai88] K.Y. Lai, T. Malone, K-C Yu, "Object-Lens: A Spreadsheet for Cooperative Work", ACM Transactions on Office Information Systems, vol 6, no 4, 1988, pp. 332-353.

[Myer86] Myers, B., Buxton, W., "Creating Highly-Interactive Graphical User Interfaces by Demonstration", *Computer Graphics*, v20, n4, August 1986, pp. 249-258.

[Myer87] B.A Myers, Creating User Interfaces by Demonstration, *University of Toronto Computer Systems Research Institute Tech. Report*, May 1987. CSRI-196 (Ph.D. Thesis).

[Meyr89] Myers, B., Vander Zanden, B., Dannenberg, R., "Creating Graphical Interactive Application Objects by Demonstration", *Proceedings of the ACM Symposium on User Interface Software and Technology*, November 1989, pp. 95-104.

[Sing88] G. Singh, M. Green, "Designing the Designer's Interface", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, Banff, Alberta, Canada, October 1988, pp. 109-116.

[Sing89] G. Singh, M. Green, "Chisel: A System for Creating Highly Interactive Screen Layouts", *Proceedings of the ACM Symposium on User Interface Software and Technology*, November 1989, pp. 86-94.

[Wiec90] C. Wiecha,, S. Boies, "Generating User Interfaces: Principles and Use of ITS Style Rules", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, November 1990, pp. 21-30.