A Domain Description Language for Sketch Recognition

Tracy Hammond, Randall Davis

The Problem: Pervasive environments, complete with digital whiteboards and pocket PC's, have increasingly included applications with sketchable interfaces. Sketch recognition applications built for the Oxygen platform include Ligature [4], Tahuti [6], and Assist [1] / Assistance [9]. To date, sketch recognition systems have been domain-specific, with the recognition details of the domain hard-coded into the system. A domain-independent recognition system is advantageous since it may be used for several domains, increasing the flexibility and capabilities of a system. However, the system cannot identify the domain shapes if it doesn't know that they are. In order to properly recognize a sketch of a particular domain, domain-specific information must be supplied to the domain-independent recognition system.

Motivation: We propose a domain description language used to describe domain-specific information to a domain-independent sketch recognition system. The language is primarily based on shape to ensure correlation between the drawn shape and the recognized shapes. and to enable designers to draw the shapes as they would naturally. The language is different from other such languages because it can be also be to describe non-shape information, including display information, editing behavior, and drawing order.

Previous Work: Shape description languages have been around for a long time [10]. These grammars have been studied widely within the field or architecture, and many systems are still built using shape grammars [5]. However, they have been developed for design generation rather than recognition, and don't provide for non-graphical information, such as stroke order, that may be helpful in recognition.

Within the field of sketch recognition, there have been other attempts to create shape languages for sketch recognition. [8] use a language to model and recognize stick figures. The language currently is not hierarchical, making large objects cumbersome to describe. [3] use fuzzy relational grammars and [2] use BNF grammars to describe shape information. Both lack the ability to describe non-shape domain information such as stroke order or direction and editing behavior information.

Approach: The difficulties in determining the language's components and syntax include ensuring that the language allows all common helpful domain information to be specified. The language must also encourage and facilitate the creation of correct programs. For instance, to encourage the reuse of geometric shape definitions, the language distinguishes between geometric shape definitions (shapes usable in many domains) and domain shapes (shapes specific to a domain). The language also provides abstract shape definitions that describe a class of similar shapes to prevent rewriting of identical attributes.

The language consists of pre-defined shapes, constraints, editing behaviors, as well as a syntax for combining them. A domain description is specified by 1) a list of the shapes and shape compositions in the domain, 2) shape definitions, 3) domain shape definitions, 4) abstract shape definitions, 5) domain shape composition definitions (how shapes interact), 6) constraint definitions, and 7) editing behavior definitions.

A shape definition describes shapes usable in multiple domains. A shape definition is composed of seven components. The *description* (line 1 in Figure 1) is a textual description of the shape. The *is-a* section (line 2) is an indication of any class of abstract shapes that it belongs to. The *components* (line 3) include the geometrical shapes of which this shape is composed. (Shapes are defined hierarchically.) Note that the TriangleArrow is composed of a pre-defined shape Line as well a user-defined shape OpenArrow. The *constraints* (line 4) specify the necessary relationships or hard constraints. They can also specify soft constraints that may not always occur in the ideally drawn shape and are thus may not be seen in the drawn object. However, these soft constraints typically occur often enough to be useful to the recognition process. If a constraint is soft, the word *importance* precedes the constraint followed by a qualifier (*high, low, medium*). For instance, a probable drawing order may be shaft, head1, l, head2.

The *derived properties* (line 5) allows us to compute certain properties and name them for use later. The *display* section (line 6) defines what should be displayed on the screen. The default is the original strokes. Generally, the original strokes are shown for all geometrical shapes, and the display changed only for domain shapes. *Editing behaviors* (line 7) can be defined for each shape. The editing behavior below allows the user to move the entire arrow by clicking and dragging the shaft. The user can also click and drag the head or tail of the arrow while the opposite end remains fixed; the shaft stretches and rotates as appropriate.

Figure 1: Shape Definition for a Triangle Arrow.

```
(define sketch-shape TriangleArrow
  (description "An arrow with a triangle head")                                %1
  (is-a Arrow)                                                                  %2
  (components (OpenArrow oa) (Line l))                                          %3
  (constraints                                                                  %4
    (meet l.p1 oa.head1.p1) (meet l.p2 oa.head2.p1)
    (angle oa.shaft l 90) (angle l oa.head1 45) (angle l oa.head2 45)
    (importance medium draw-order oa.shaft oa.head1 l oa.head2))
  (derived-properties                                                           %5
    (Point head oa.shaft.p2) (Point tail oa.shaft.p2)
    (Line shaft oa.shaft) (Line head1 oa.head1) (Line head2 oa.head2))
  (display (cleaned_strokes shaft)(ideal_strokes l head1 head2)                 %6
  (editing-behavior                                                             %7
    (click_hold_drag head (fix tail) (stretch_scale_rotate this) (move head))
    (click_hold_drag tail (fix head) (stretch_scale_rotate this) (move head))
    (click_hold_drag shaft (move this))
    (scribble shaft (delete this))))
```

Impact: We present a language for describing domain-specific information to a domain-independent sketch recognition system. By describing domain descriptions using the language's syntax, designers can add sketch recognition to their user interfaces without coding sketch-recognition details into their system.

Future Work: In the future, we will test human usability by asking users to develop domain descriptions using the proposed language. We will test that these descriptions agree with the users' intensions by developing a simple domain-independent sketch recognition system.

Research Support: The language described in this paper is part of a larger project [7] that includes learning shape descriptions, compiling shape descriptions, and recognizing sketches based on these descriptions. This work is being done by members of the Design Rationale Group at MIT, lead by Randall Davis, and include Christine Alvarado, Tracy Hammond, Michael Oltmans, Metin Sezgin, and Olya Veselova. This work is supported by the MIT Project Oxygen partnership and by DARPA through the Office of Naval Research under contract number N66001-99-2-891702.

References:

[1] Christine Alvarado. A natural sketching environmant: Bringing the computer into early stages of mechanical design. Master's thesis, MIT, 2000.

[2] Oliver Bimber, L.M.Encarnao, and Andre Stork. A multi-layered architecture for sketch-based interaction within virtual environments. *Computer and Graphics*, 2000.

[3] Anabela Caetano, Neri Goulart, Manuel Fonseca, and Joaquim Jorge. Javasketchit: Issues in sketching the look of user interfaces. *AAAI Spring Symposium on Sketch Understanding*, 2002.

[4] Mark Foltz. Ligature, gesture-based configuration of the e21 intelligent environment. *MIT Student Oxygen Workshop*, 2001.

[5] James Gips. Computer implementation of shape grammars. *NSF/MIT Workshop on Shape Computation*, 1999.

[6] Tracy Hammond and Randall Davis. Tahuti:a geometrical sketch recognition system for uml class diagrams. *AAAI Spring Symposium on Sketch Understanding*, pages 59–68, March 25-27 2002.

[7] Tracy Hammond, Metin Sezgin, Olya Veselova, Aaron Adler, Michael Oltmans, Christine Alvarado, and Rebecca Hitchcock. Multi-domain sketch recognition. *MIT Student Oxygen Workshop*, 2002.

[8] James V. Mahoney and Markus P. J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. *AAAI Spring Symposium on Sketch Understanding*, pages 105–112, March 25-27 2002.

[9] Michael Oltmans. Understanding naturally conveyed explanations of device behavior. Master's thesis, MIT, 2000.

[10] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. *Information Processing*, pages 1460–1465, 1972.