Implementation of a Constraint-Based Visualization System

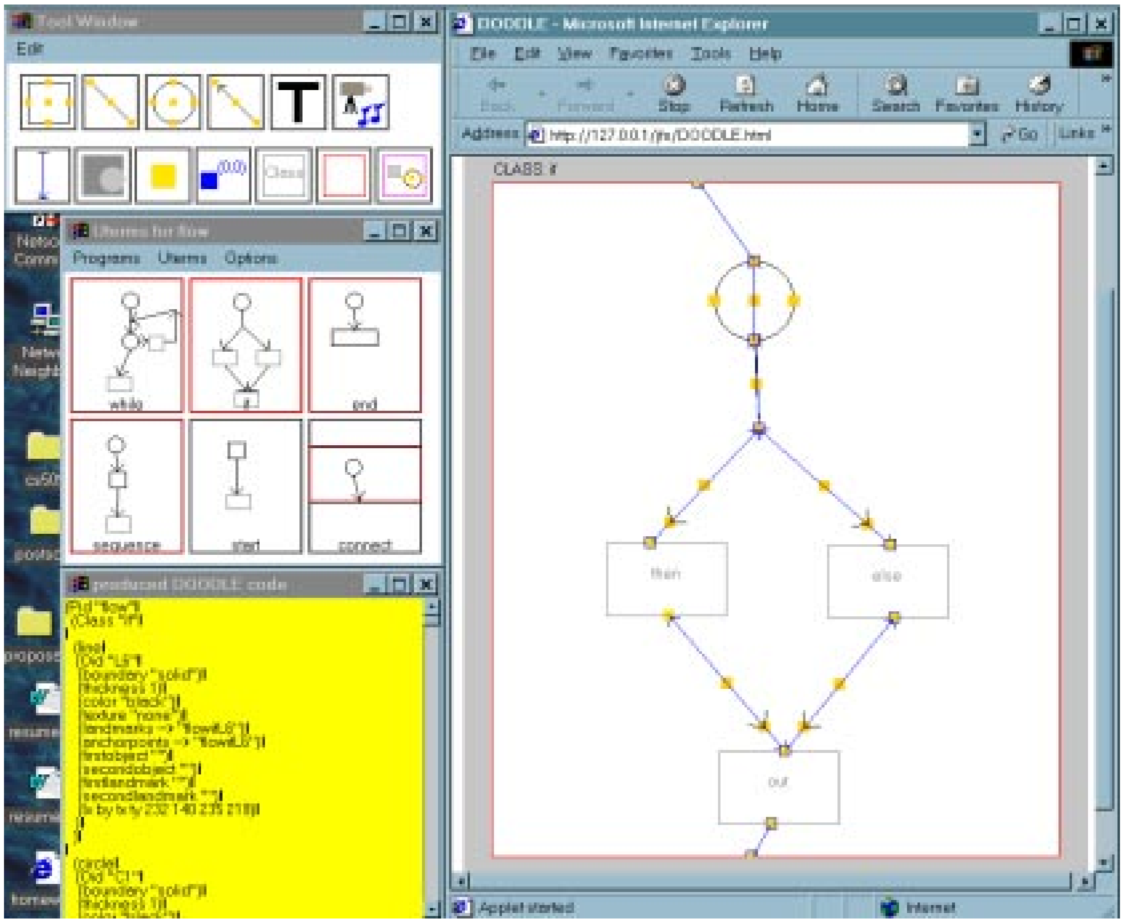
Isabel F. Cruz
Peter S. Leveille

Information visualization focuses on graphical mechanisms designed to show the structure of information and improve the cost of access to large data repositories. Declarative approaches to information visualization allow the user to specify what the displays look like but not how they are to be produced from the specification. While declarative systems have been developed to automate the designs of graphs and diagrams, oftentimes they depend upon textual specifications and on time-consuming constraint solvers, e.g., based on simulated annealing or genetic algorithms.

Our approach takes advantage of the old adage "a picture is worth a thousand words," and uses a visual language for the specification of the visualizations with which users can specify their own visualizations. To support this visual mode of interaction, a sophisticated system called Delaunay was built that accepts both the user's specification and the information in the database, and produces images that depict that information.

Advanced visual interfaces have been implemented for the input of the visual specifications and for the output of the generated visualizations in Delaunay. The input interface is shown Figure 1. In this example, the user is specifying the visualization of a flowchart. In the drawing pad on the right, the user assembles visual symbols from the top left portion of the screen. In this case, the *if* statement in a flow chart is being specified. The previously specified components of the flowchart are shown to the user as interactive thumbnails on the left of the drawing pad.

The flowchart produced by this specification for one of the flowcharts in our database is shown in Figure 2, which also shows the output component of the user interface. In this component, visualizations can be manipulated using panning, zooming, and fish-eye views. In addition, the interface allows for the textual data associated with each graphical object to be inspected on demand. These features are shown in Figure 3 that displays an n-ary tree as visualized using Delaunay.



```
(Fid "flow")
 (Class "if")
(
 (line)
 (Oid "L5")
  (boundary "solid")
  (thickness 1)
  (color "black")
  (texture "none")
  (landmarks -> "flowL5")
  (anchorpoints -> "flowL5")
  (firstobject "")
  (secondobject "")
  (firstlandmark "")
  (secondlandmark "")
  (lx by tx ty 232 140 235 218)
 )
 )
(
 (circle)
 (Oid "C1")
  (boundary "solid")
  (thickness 1)
```

Figure 1: Specification of the *if* component of a flowchart.

The figure also illustrates the fisheye view feature of the output with different distortions and the exploration of the data associated with a visual object (shown on the right panel).

In order to allow for a powerful and flexible tool, most of the specification is left to the user. However, when appropriate, the system will enforce certain rules of good visualization by judicious encoding of data attributes using visual attributes. For example, hue is used for encoding *nominal* data (i.e., data without quantitative content) instead of color saturation, another possibility offered by the system. In addition, it is also important that Delaunay be "intelligent" about choices that users leaves to it. For instance, if we tell the system that we wish to have random colors assigned to a set of objects, it should (1) assign colors to

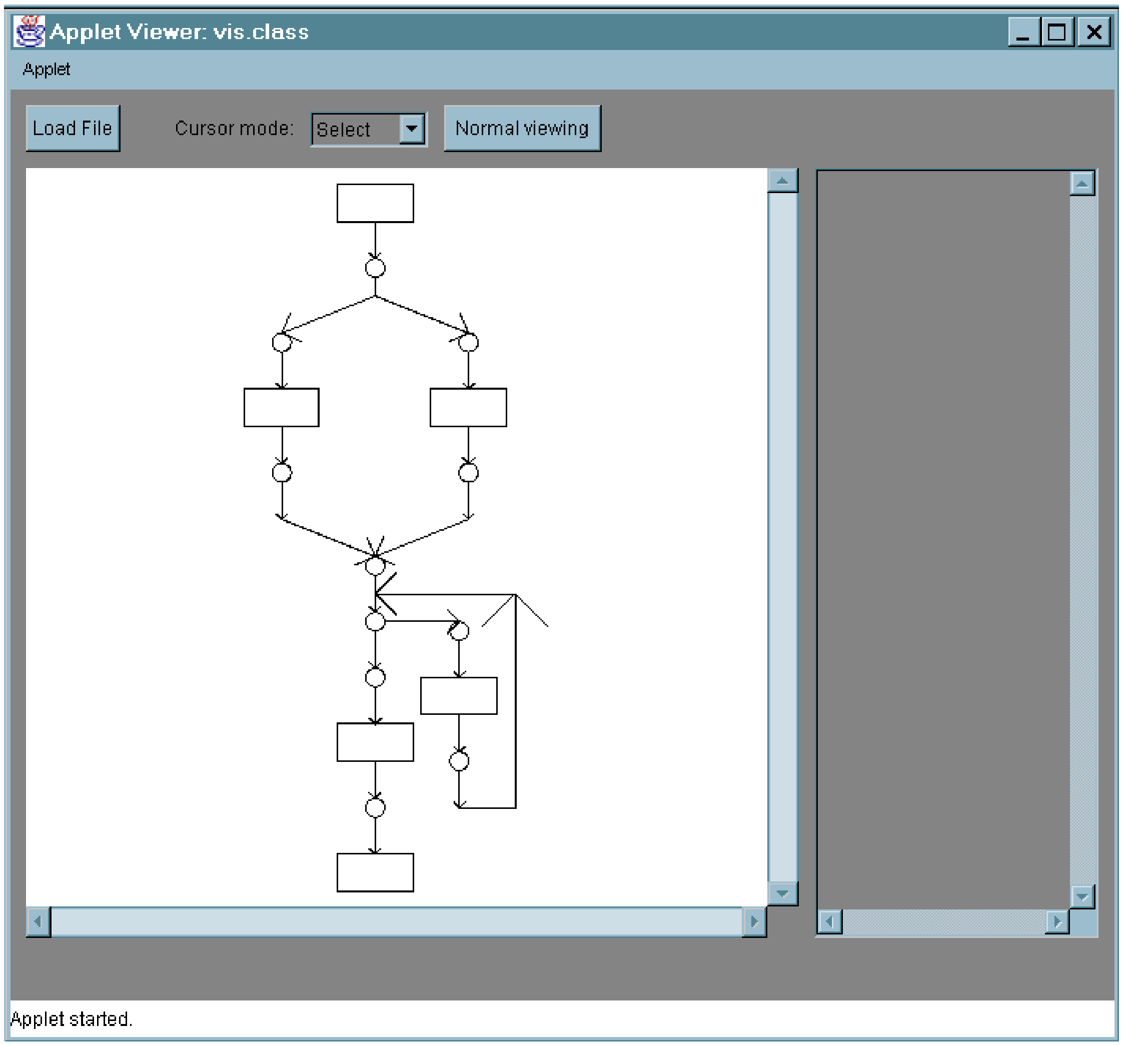


Figure 2: Visualization of a flowchart.

each object that are sufficiently different from each other; (2) take into account other colors previously assigned in the specification in order not to overload the meaning of that color.

Automatic assignment of colors was performed in the example of Figure 4. In this relational diagram, we chose to represent the relationship between Australian animals and traits that occur in these animals. Multimedia objects are used to display both. Lines express the relationship between each animal and a trait

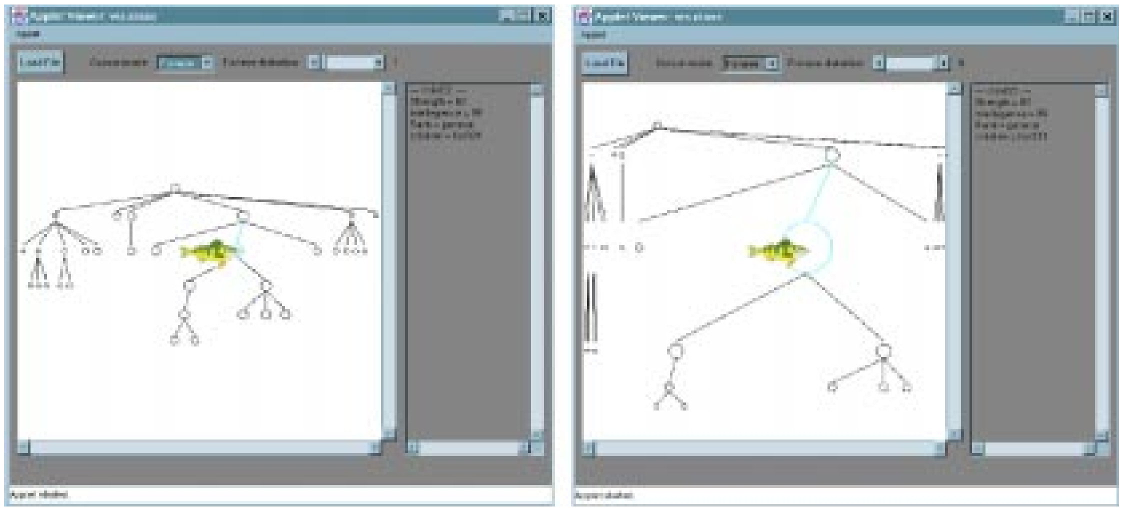
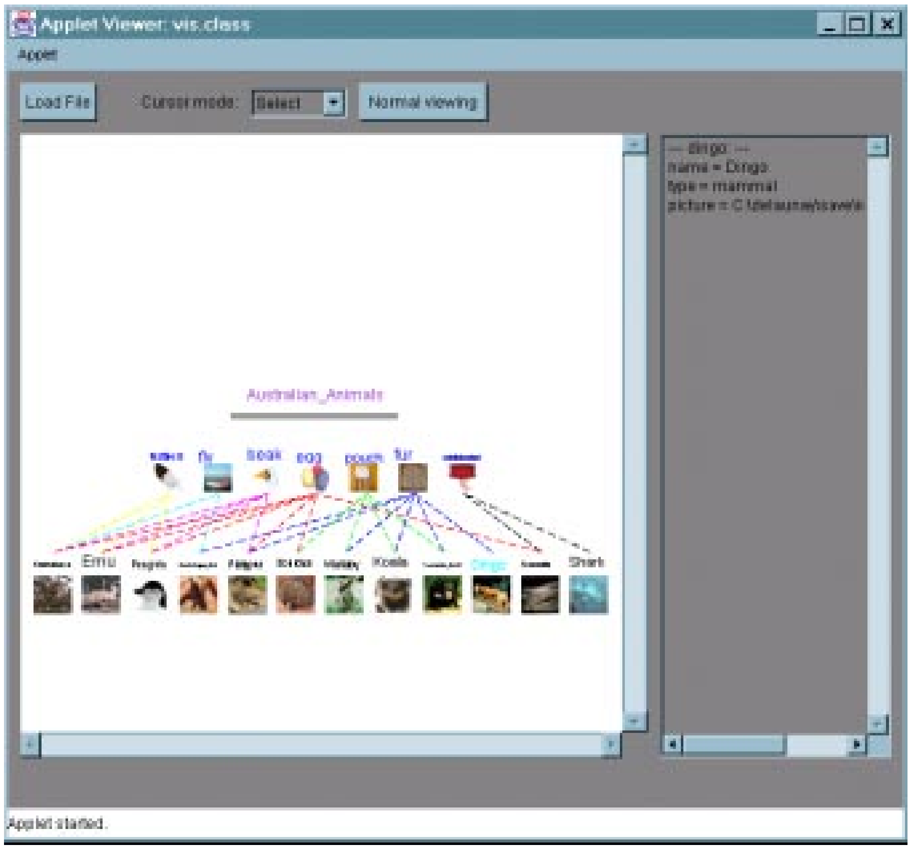


Figure 3: Two different fisheye distortions are illustrated; an actual fish was superimposed on this picture to show the focus of the view.

that the animal possesses. The user just has to specify that the lines emerging from the same trait should have the same color encoding, and the system automatically chooses the different colors, obeying the above rules. Since the data being displayed is nominal, the system chooses color hue for the representation.

To determine the actual positions of the visual objects on the output, our system uses constraint satisfaction techniques. In fact, a key component of the system is the constraint solver. Constraint solving is not a trivial issue. Previously, the SkyBlue constraint solver (the result of a PhD thesis at the University of Washington) had been used in Delaunay. Unfortunately, SkyBlue cannot solve cyclic constraints that are common in practice. Therefore, we implemented a new constraint solver, capable of handling efficiently the constraints appearing in Delaunay.

Delaunay uses a class of basic constraints over a set of variables, where linear arithmetic expressions are combined with min/max operators. The variables represent the values of other constraints in the same specification or the value of a database attribute (so that we can set a length to be proportional to that attribute value). The constraint solver runs in quadratic time in the number of equations that are related to each variable. We found that a small number of variables is needed even in the most complex visual specifications, and that the number of equations associated with a variable is also small. Therefore, in practice, the constraint solver is efficient.



```
--- dingo ---
name = Dingo
type = mammal
picture = C:\delaunay\save\
```

Figure 4: A relational diagram.

The Delaunay system has been developed and tested with a rich variety of visualizations, proving that it is a powerful and useful visualization system. Examples of visualizations created by Delaunay include the drawings of graphs (trees, series-parallel graphs, bipartite graphs) quantitative and temporal charts, relational charts, and X-Y plots. Contrary to other systems such visualizations are not predefined. Instead, they are just examples of a potentially unlimited number of visualizations that a user can define, for any domain of choice. For example, in business analysis, diverse visualizations such as stock charts, Gantt diagrams, and flowcharts have been deployed.