Report Number: WUCS-01-44

2001-01-01

Legends as a Device for Interacting with Visualizations

Mihail E. Tudoreanu and Eileen Kraemer

Users and developers of visualization tools must deal with the problem of specifying what information to show and how to represent it. Typically, the user's focus of interest will change over time, and the specifications must change with the user's interests. Techniques for the simple, direct, and intuitive creation and refinement of these specifications can be useful. In this paper we show how legends, a natural element of graphical displays, may be used as a direct and unobstrusive interaction device through which users may interactively specify new visualizations and animations.
... Read complete abstract on page 2.

Legends as a Device for Interacting with Visualizations

Mihail E. Tudoreanu and Eileen Kraemer

Complete Abstract:

Users and developers of visualization tools must deal with the problem of specifying what information to show and how to represent it. Typically, the user's focus of interest will change over time, and the specifications must change with the user's interests. Techniques for the simple, direct, and intuitive creation and refinement of these specifications can be useful. In this paper we show how legends, a natural element of graphical displays, may be used as a direct and unobstrusive interaction device through which users may interactively specify new visualizations and animations.

Legends as a Device for Interacting with Visualizations

Mihail E. Tudoreanu and Eileen Kraemer

WUCS-01-44

December 2001

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

Legends as a Device for Interacting with Visualizations

Mihail E. Tudoreanu
Department of Computer Science
Washington University
St. Louis, MO 63130

Eileen Kraemer
Department of Computer Science
The University of Georgia
Athens, GA 30606

ABSTRACT
Users and developers of visualization tools must deal with the problem of specifying what information to show and how to represent it. Typically, the user's focus of interest will change over time, and the specifications must change with the user's interests. Techniques for the simple, direct, and intuitive creation and refinement of these specifications can be useful. In this paper we show how legends, a natural element of graphical displays, may be used as a direct and unobtrusive interaction device through which users may interactively specify new visualizations and animations.

Keywords
Visualization, direct manipulation, legend, interaction device.

INTRODUCTION
Visualization, the use of images to convey meaningful information, encompasses sub-fields such as scientific, information, process, and software visualization. In each of these types of applications, the user of a visualization tool may be faced with the problem of what to show, how to show it, and where to start. The developers of these tools must face the problem of how best to permit the user to specify their initial interest and to then refine these specifications. In this paper we present legends as an interactive device through which users may interact with visualizations to dynamically alter the specification of what data to collect and how to present it.

Legends are appealing because of their familiarity and simplicity, and the potential for direct and natural manipulations. Legends are found on many maps and graphical displays. Automatic data presentation tools such as APT[1] and SAGE[3] include legends in the visualizations they generate. Through repeated exposure to these displays, most people understand legends and can divine their meaning for a given display. Further, legends are simple, a mapping of pictorial elements to data. Interactions with legends may also prove to be simple. Thus, this kind of interaction may prove suitable for computing devices that are windowless or have small displays, such as PDAs. Legends provide a way of adjusting the appearance of visualizations by acting upon the visualization itself. There is no need to switch to other windows or tools, and users avoid the process of iteratively altering an indirect representation, invoking a compiler, and then switching to the picture to check the result. Instead, interaction with the visualization itself produces an immediately observable change that can then be reversed, continued, or otherwise modified, a style of interaction is well-suited for design and exploration tasks. Legend interaction can be integrated with other tools that facilitate interactions with visualizations, like Visage[2], a system that supports manipulation of data across multiple visualizations displays.

Legends can evolve along with the data set they represent, and in this way provide the user with an appropriate set of interaction choices for the current state. Animation characteristics can also be specified, through legends in which the temporal dimension and its mapping to graphical attributes is made explicit. As a motivating example we consider a scenario in which a user, through visualization and interaction, is exploring the behavior of a distributed genetic algorithm.

Name	Field
Individual	Gene
	Fitness
Deme	deme id
	Size
	array of individuals
	(row, column) of the grid
member(individual, deme)	
send(from, to, individual)	
recv(from, to, individual)	

Table 1: Variable types and relation.

INTERACTION STYLE
In this section we illustrate the types of manipulations users may perform through legends and attempt to give the reader an idea of the gestural language available to users. We present a sample visualization session, and describe how a hypothetical user analyzes the execution of a distributed genetic algorithm. A genetic algorithm is used to approximate the maximum (or minimum) value of a cost

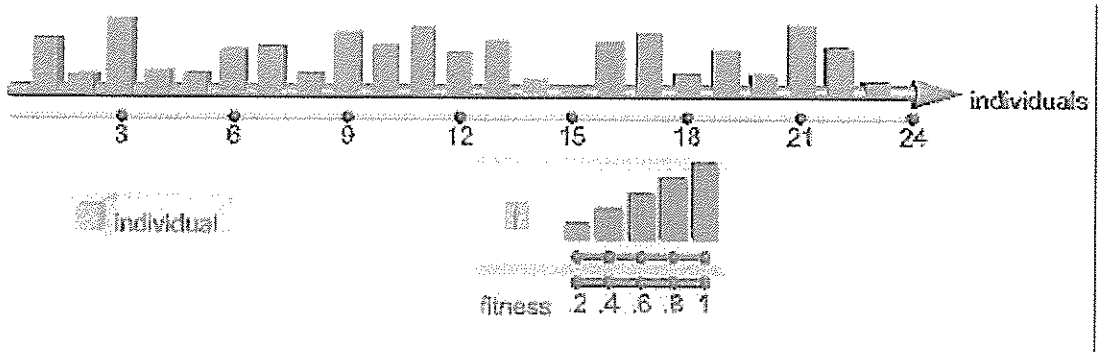

Figure 1: The fitness of all individuals.

function and to obtain a solution, a member of the function's domain that yields the approximated maximum. A number of potential solutions, called *chromosomes* or *individuals*, are distributed at the nodes of a network, in this case with a grid-like topology. A group of individuals local to a node form a *deme*. From time to time, individuals are exchanged (migrate) between adjacent demes. Each node processes its deme in steps known as generations. The population of one generation is obtained from the previous generation through a process similar to natural selection: individuals that are most *fit*, as determined by the cost function, are most likely to pass on their encoding to the next generation.

The visualization system produces displays based on the state of the computation. With each change in state in the computation, the visualization system updates the display. The system has knowledge of the type of variables in the state as well as the relationships between them. This knowledge is either embedded in the computation code by a programmer or extracted by code analysis or instrumentation tools. Variables and relations for the genetic algorithm are presented in Table 1. The member relation identifies the deme each individual belongs to, while send and recv correspond to the sending and receiving demes of a migrating individual.

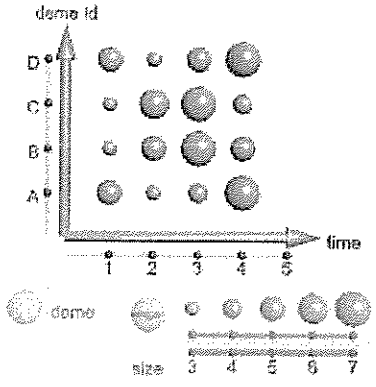

Figure 2: The evolution of deme sizes.

Initially, the visualization system constructs, either automatically or with user's guidance, the four `simple' visualizations of Figure 1, Figure 2, Figure 3 and Figure 9. Each visualization has a legend that provides insight into the meaning of the graphical elements of the picture. A legend consists of keys, each describing a specific visual element. The legend in Figure 2 has four keys: one conveys that a sphere represents a deme. Another conveys the relation between the radius of a sphere and the deme size. The coordinate axes, useful in understanding the correspondence between the data they encode and the location of elements in the picture, are also considered keys.

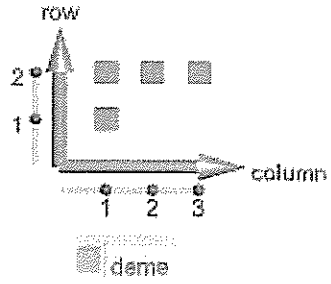

Figure 3: The position of the demes on the network grid.

Keys can be classified in two categories based on the type of information they present. One category of keys, *object-entity keys,* shows how entities of the program, such as demes or individuals, are related to the types of graphical objects in the picture. Such keys appear in the visualization as a rectangle that holds an image of the graphical object and the name of the entity (left on Figure 2). The other category of keys, *attribute-scale keys,* show how properties of data entities are encoded visually with attributes of their graphical object. These keys (right on Figure 2) depict the graphical object and attribute and the name of the data property and show a list of graphical values and the corresponding list of data values of the property. The values appear as dots, or *ticks*, on a thread. Special representations are assigned to keys describing textual labels and to coordinate axes. Textual labels do not have the two lists of values nor the threads. Coordinate axes consist of an arrow with ticks, the data values corresponding to these marks, and the name of the data attribute.

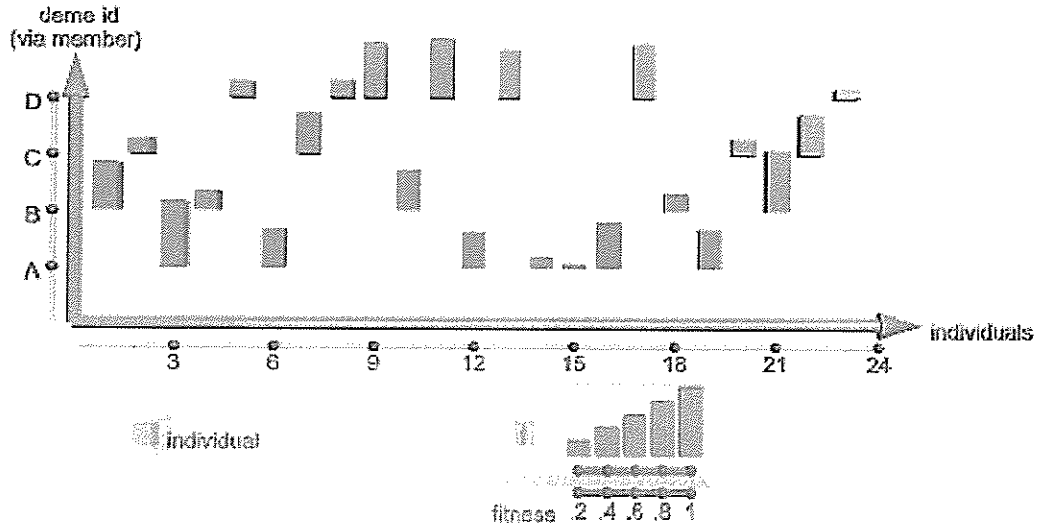

Figure 4: Individuals are moved to the y-position of their deme id.

Typically, the ticks, values of visual attributes and data that are displayed in a key, are automatically chosen at a fixed distance from one another based on the range of data and graphical values of each thread. The range is determined by the minimum and maximum values of the elements displayed in the picture. If a small number of values are encoded by a key, all values appear as ticks. If too many values need to be displayed, the key shows only a few samples. The tick marks corresponding to selected elements appear when the user selects graphical objects in the visualization.

We now describe a scenario in which the user concentrates on the visualization of Figure 1. The user will interact with the legends of the visualizations to group individuals by deme, to move the objects corresponding to individuals to positions based on deme, and to animate the migration of individuals between demes. For simplicity of presentation in this paper, the number of individuals was reduced.

First, the user wishes to mark each individual depending on the deme they belong to. To do so, the deme id key, the y-axis, is dragged from Figure 2. A dragging gesture copies selected elements to a target and is interpreted as a request to modify the target. In this case the y-axis of Figure 2 is dropped on the visualization of Figure 1 with the result that the visualization receives not only the graphical attribute of the key but also the data values that already exist on the y-axis, deme ids A, B and C. The graphical objects of the picture now exhibit a new visual property. The system is able to associate individuals to deme-ids through the member relation, and thus to y-locations. The result is that the bar object of each individual now has its bottom aligned with the y-position corresponding to its deme. The member relation, which was used by the system to associate deme ids and individuals, becomes part the name of the key: "deme id (via member)" (Figure 4). If additional deme-individual relations were defined, the user could cycle through the available relations by clicking on the "member" label. After a click, the visualization would change to reflect the current relation.

The current visualization (Figure 4) has individuals of the same deme scattered along the x-axis. The user decides that it is better to have them grouped by deme. This can be done by sorting their x-axis values on the deme id. To do so, the user drags the vertical coordinate, which is the key that encodes the deme id, to the x-axis. As a result, the x-axis becomes a key that encodes the same data values as before (e.g. individuals 1 through 23), but in an order specified first by the deme id, then by the individual number. In general, this type of operation leaves unchanged a key whose graphical attribute is inherently unordered for human perception such as text, shape and texture. However, the label of the new key is always changed to show the attribute on which it is ordered, as shown in Figure 5.

In the remainder of the scenario, the user wishes to put the individuals at their actual positions on the map and animate their migration. To achieve this, the user drags the coordinates of the map (Figure 3) to the current visualization (Figure 5). A conflict occurs, as the visualization already has x and y axes. The system highlights the keys involved to bring the conflict to the user's attention. Conflicts can be resolved by eliminating one of the conflicting keys or by changing its graphical attribute. In our scenario, the user chooses to eliminate the former x-axis and to change the deme id encoding to color. Individuals are thus moved to the map position that corresponds to their demes (via member relation). The user

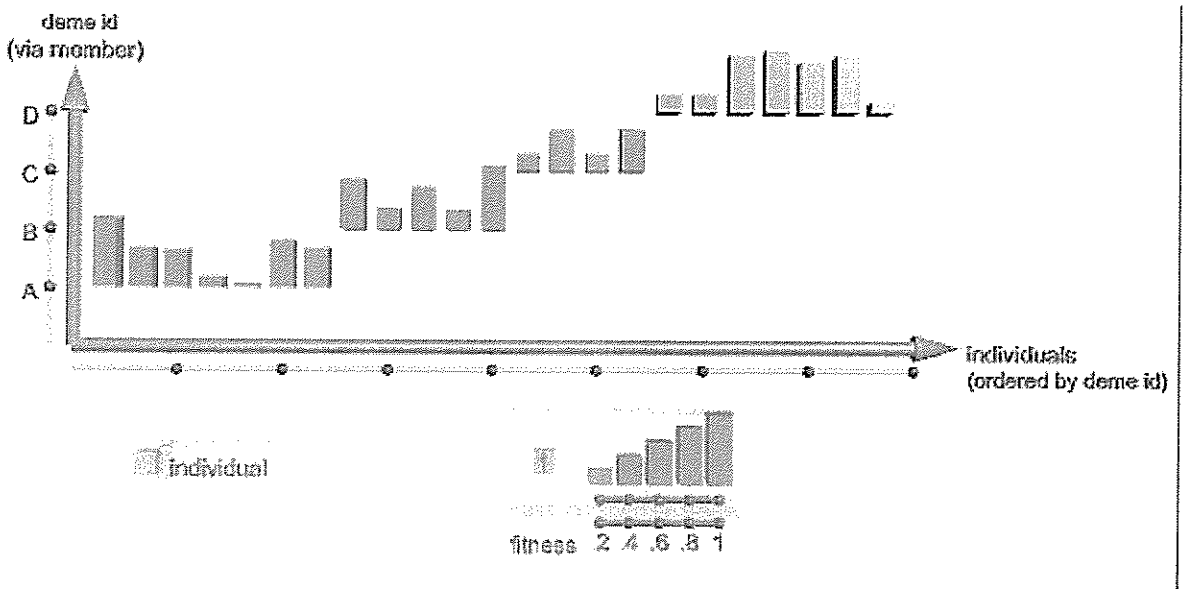

Figure 5: The x-axis is ordered by deme id.

can cycle through the available graphical attributes for a key by clicking the representation of the current attribute, or the arrow in the case of a coordinate. One of the possible values is an empty graphical attribute which has the effect of deleting the key from the visualization. The deletion is promptly reflected in the visualization, but the key remains for a while to give the user a chance to continue moving through the available graphical attributes.

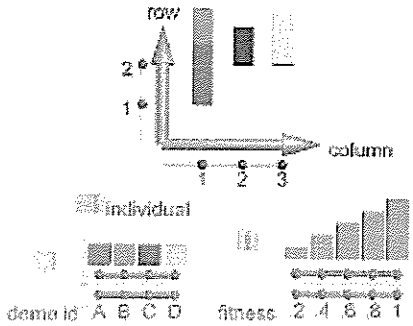

Figure 6: Individuals of a deme are overdrawn at the same grid position.

At this point, all individuals of a deme are drawn on top of each other, at the same x-y location (Figure 6). This happens because the typical behavior of the visualization is to map one data value to a single graphical value. However, the user has the option to assign a range of graphical values, henceforth termed a *band*, to the same data value. Graphical objects in a band have the same data value, but are automatically spread out so they can be individually examined by the user. For this reason, the creation of a band can be thought of as the 'opening' of an attribute value. The gesture is a double click on the tick that corresponds to the value to be opened. The dot is replaced by a cylinder whose width is chosen by the system with the goal that all graphical objects with the specified value are distinguishable. However, if a large number of objects with the opened value exists, the system assigns a smaller band than needed, with the result that the objects may still partly overlap. The user can then adjust the value of a band by pulling its ends. The spacing between bands is preserved.

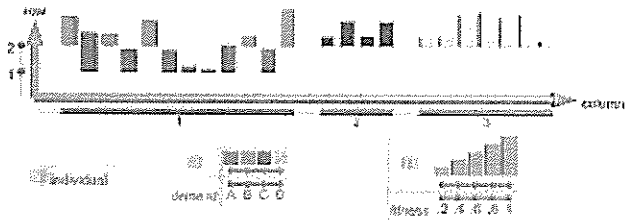

Figure 7: Individual at the same column are distinguished upon.

In our scenario, the user next selects a bar object, which results in that individual's x-location being shown on the x-axis as a mark. The user then double clicks on that mark to separate the objects at that position. After this operation, all the ticks on the data thread of that key become highlighted for a short period of time. The user clicks on one of the highlighted elements to request that the system perform the same operation for all values of the graphical attribute, with the result that objects no longer overlap on the x-axis (Figure 7).

As an aside, we note that a user may choose to aggregate the elements of a band into a single element. This is accomplished by moving one end of a band on top of the other and has the result that one bar object appears in the visualization for each x-position. The other attributes of the

bar are also aggregated. For numeric attributes such as position, height or radius, the value of the aggregate is the average or the sum of the attributes of the components. For colors and textures, the system can present each of the colors and textures in a portion of the aggregated object. Shapes typically overwrite one another. When a small number of elements is involved, text labels can be aggregated using the first character of the label of each component.

Returning to our scenario, we see that one problem with the current visualization is that individuals are too close together in both the x and y dimensions. The height, which encodes the fitness, causes the individuals to overlap due to lack of space on the y-axis. On the x-axis the user finds it difficult to distinguish between members of adjacent demes. Both problems require that the spacing between positional values be increased. The user accomplishes this by 'pulling' the end of each axis until a satisfactory spacing is achieved (Figure 8).

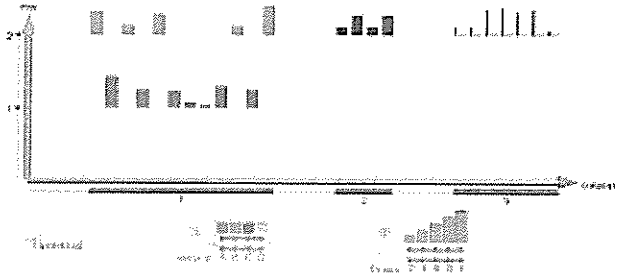

Figure 8: Grid positions are spaced from each other.

In general, the values of a key or coordinate axis can be regarded as marks on a rubber/elastic thread. Increasing the length of the thread has the effect of increasing the relative distance between marks. Most keys have two threads, one for graphical values and one for data values. A data value corresponds to the graphical value directly above it. In the scenario, by 'stretching' the data threads of both axes, the user controls the distance between members of different demes. A thread can be pulled from either an end-point or an inside point. If pulled from an inside point, the end-points remain fixed and consequently one part of the thread widens and the other shrinks and the linearity of the mapping of data to graphics is altered. Note that a band is considered a single mark, so its size does not change through this interaction. The size of the band can be modified by manipulating its ends.

Animation

Time provides another dimension through which information can be presented. Thus, the manner in which a graphical change is animated is another attribute for which a key may be included in the legend. Animation keys have a similar appearance to graphical attribute keys and are subject to the same manipulation gestures. However, they are conceptually different from graphical keys, and the semantics of some operations differ.

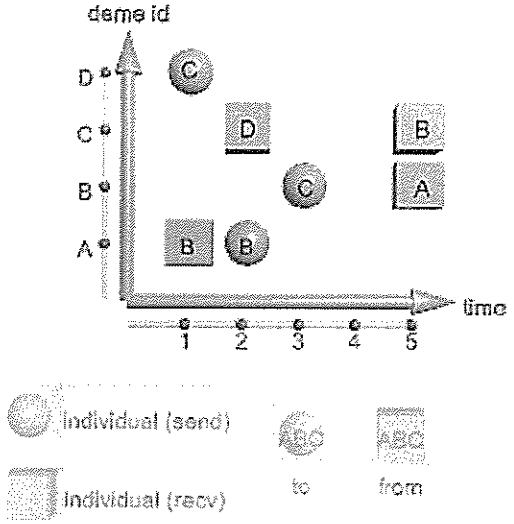

Figure 9: Sending and receiving individuals between demes.

Animation is dependent on logical time, determined by the evolution of the program, and incremented when the state changes. Logical time is a key that appears in all visualizations either explicitly, as in Figure 9, or implicitly as in Figure 8. In the latter case, time can be considered as appearing on a z-axis perpendicular to the screen. Frames depicting previous states can be regarded as a stack, with the most recent frame visible. The time key resembles the controls of a VCR with controls that start, stop or step through the process of time advance.

Animation is performed by smoothly varying one or more attributes of graphical objects over a period of time. Attributes are assumed to have one dimension imposed by a total ordering of their values. Initial, final and possibly intermediate values for each attribute are specified. These values, termed *interpolation points*, are the values of the attribute at given logical time points; i.e., in certain states. The attribute is interpolated from the previous point to the current point in real time. The interpolation is not necessarily linear; speed may vary during interpolation.

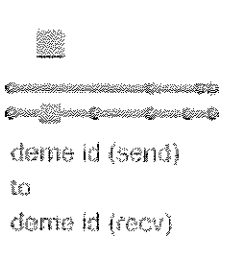

Figure 10: Example of animation key

An example of an animation key is given in Figure 10. In the upper region, the type of object involved in the animation is presented. A sample object is animated in an infinite loop, starting with an initial value of the attribute, and smoothly changing to a final value. The object briefly stops at each interpolation point. The values of the interpolation points for the sample object are chosen by the system based on the type of graphical object and animated attribute.

In the middle region, the interpolation points and the relative distance between them are shown by means of a thread. A second thread is used to display the duration and the speed at the interpolation points. The tick marks on the upper thread indicate interpolation points. A user may interact to add or remove points by clicking on the desired location of the thread. The lower thread represents time and, in addition to tick marks, has a slider that moves from one end of the thread to another. The slider takes one second from one tick to the next, independent of the spacing between ticks. The slider movement changes the appearance of the object(s) in the upper region of the key. At any given time the value of the animated attribute is the value shown above the slider. So, the closer the tick marks on the time thread, the lower the animation speed. In Figure 10, it takes 2 seconds to move from the initial interpolation point to the next one. In the first second, the individual moves quickly, but in the next the individual moves more slowly.

The lower region is used to depict the keys that supply values for an interpolation point. Only two points need to be defined; the value of others can be inferred. It is practical to require that the final point be explicitly specified before the animation of an object ends. If the end point is not known for an object, that object does not perform the last two interpolations. This permits the specification of the final value in a later state, after the object has started the animation. The second to last value serves as a temporary destination.

To create an animation key, the user selects the objects to be animated from an existing key and the attributes that will change from other keys. These keys are dropped on the time key that appears in each visualization. A new animation key is added for each attribute. By default, the interpolation has two unassigned points and the speed is constant. The user can add new points and change the time thread. Values must be assigned to at least two points for the animation to begin.

In our user's scenario, the circle objects of Figure 9 and the coordinate axes of the current visualization (Figure 8) are used to create new animation keys, with the result that the sending of individuals between demes is animated along the coordinates. For the rest of the discussion, we will focus on the animation on the x-axis. Animation of the y-axis is similar.

The user adds two more interpolation points: one past the middle of the upper thread and the other close to the end. The desired animation effect is for the individual to move more than half way towards the destination deme, but to complete the movement only after the destination deme has actually received the individual. The initial and the third points are set when an object starts the animation, that is, when it is sent. The second point is automatically computed, while the last point will be set when the object is received.

To specify the starting point, the user selects both the y-axis of Figure 9 and the key that associates spheres with sends. In this manner, only the deme ids corresponding to send events are considered. The selection is then dragged and dropped on the initial interpolation point of the animation key (Figure 10). Next, the key showing the label of the circles, the actual destination, is dropped on the third point. Note that once the deme id is known, the x-position on the grid can be easily determined.

The system can now search for bar objects to be animated. The individuals associated with these objects are represented as spheres in Figure 9, as those were the objects used to create the animation key. The system takes the values of both the deme id on the y-axis and the deme id in the label of those spheres and initializes the interpolation values for the corresponding bar objects. Those objects start the animation. However, none can finish yet (not even individuals that were actually received) because the system does not know the final interpolation point.

To specify the final point, the box objects and the y-axis of Figure 9 are first selected. They are dragged to the final interpolation point. The system can now search to find bars that have already started the animation and are also boxes in the visualization of Figure 9. The ones that are found finish the animation because their final interpolation point receives the value of the deme id of the box. To prevent finishing the animation for an object that was received from one deme and then sent to another, a box is considered related to an animated bar only if it exists at a logical time later than the time corresponding to the beginning of the animation of that bar.

While new send objects appear in Figure 9 the corresponding bars begin the animation process. Similarly, new receive objects lead to the end of animation for the corresponding bar objects.

DISCUSSION

The gestural language for visualization manipulation requires a pointing device and the ability to click, or touch, and to drag components of the visualization. The components that can be manipulated, the available gestures, and the semantics of operations are presented in Table 2. After an operation is performed for a tick or band, the key

Component	Gesture	Semantics
Graphical object	Selection	Ticks and bands corresponding to the selected objects are shown
Ticks	Double click	Changes to a band
Band	Double click	Changes to a tick
	Move end points	Modifies the range of graphical values associated to the band. If range is zero, request aggregation.
Thread	Grab and pull on a point	Stretches or shrinks the thread or parts of it. Applicable to all threads except interpolation thread.
Coordinate axis	Click	Displayed as a regular thread
Animation key threads	Click	Adds or removes ticks.
Attribute of a key	Click	Cycles through available graphical attributes.
Data label of a key	Click	Cycles to the next available data attribute.
Relation label of key	Click	Cycles to the next available relation.
Key	Drop on a visualization	Added to that visualization.
Attribute-scale keys	Dropped on a graphical key	Orders the target key if possible.
	Dropped on temporal key	Creates an animation key
Entity-object key	Dropped on an attribute-scale key	Changes the graphical object of the target. All objects of the dragged type receive a new attribute

Table 2: Summary of interaction gestures

remains highlighted for a period of time to permit users to perform the same operation for all values of the key with a single gesture.

The visualization system must have knowledge of both the underlying data and of the characteristics of the graphical elements that are employed in the visualization. From the underlying data, the system needs to link properties of related entities in order to support the transfer of the keys among visualizations. Information about graphical attributes, such as the difference between values that is distinguishable for a human viewer or whether the attribute is continuous or discrete, helps the system decide what values to include in the keys. The characterization of the attributes, based on automatic data presentation techniques ([1] [3]), can be also employed to choose the order in which the available attributes are presented to user. The attribute best-suited for presenting the data property is shown first.

Limitations

Although legends can be used to refine visualizations and to obtain quite different views of the data, they are not helpful in building visualizations from scratch. An initial set of visualizations that is rich enough to ensure the reachability of other interesting visualizations must be defined.

Changing graphical objects and attributes as well as modifying the relations that associate a property of an entity to another entity are potentially 'hard to do' when the system is complex and numerous visual elements exit.

Another limitation is that interpolation points for an animated object must be associated with the object in some existing visualization so that the necessary key exists.

REFERENCES

1. Mackinlay, J.D. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics2*, 5 (April 1986), 110-141.
2. Roth, S.F et al. Visage: A User Interface Environment for Exploring Information. *Proceedings of IEEE Information Visualization* (San Francisco, October 1996), 3-12.
3. Roth, S.F., and Mattis, J. Automating the Presentation of Information. *Proceedings IEEE Conference on AI Applications,* (Miami Beach, FL, February 1991), 90-97.