Defining and Editing Constraints Graphically by Treating Constraints as Objects

Ilhwan Kwon
Brad A. Myers
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA. 15213
bam@cs.cmu.edu
http://www.cs.cmu.edu/~Amulet

ABSTRACT

GILT (**G**raphical **I**nterface **L**ayout **T**ool) is a visual graphical user interface development environment. It allows for specification of graphical object behaviors at run time. This behavioral specification is attained through direct manipulations on graphical objects and dialog boxes. GILT uses constraints which are maintained by the system, to specify run time behaviors. To establish the constraints, GILT explores a number of different interaction techniques. These include

- GILT defines the specific values of an object as handles and allows constraints to be specified by linking two handles together.
- Each constrained handle displays constraint information that can be directly edited.
- GILT treats constraints as graphical objects, which can be manipulated in the same way as other ordinary graphical objects.

This paper further describes these interaction techniques.

Keywords

Constraints, Interaction Techniques, Feedback objects, Amulet, Handles, UIMS

INTRODUCTION

When architects design a building, they sketch or build a model rather than writing descriptions of the design. This is because the graphical representation is more readily visualized than an abstract description. For the same reason, a software developer designing a graphical user interface by way of direct graphical manipulations uses more efficient process than the developer who attempts to achieve the same task by writing code. Ultimately, specification of object's runtime behaviors is best achieved when it is done graphically with an instant feedback. Although there have been a number of UIMS that address this issue, problems still remain. These systems often have complicated user interfaces and are limited to graphical presentations of constraints that can not be manipulated as objects. GILT, on the other hand, treats the constraint visualization objects as ordinary graphical objects that can be directly manipulated. By allowing the direct manipulations on the visualization objects, GILT keeps dialog box use to a minimum. This is beneficial because the dialog box pulls user's focus from the objects being manipulated, which causes a break in continuous interactions. GILT also provides an instant feedback when constraints are specified.

GILT is an interface builder for Amulet[1], and stands for **G**raphical **I**nterface **L**ayout **T**ool. Amulet is a user interface development tool kit for C++. Each Amulet object stores its properties in *slot*s, and constraints are used to connect values of slots together. Amulet uses a one-way constraint system, which means that value of one slot can depend on another slot, and

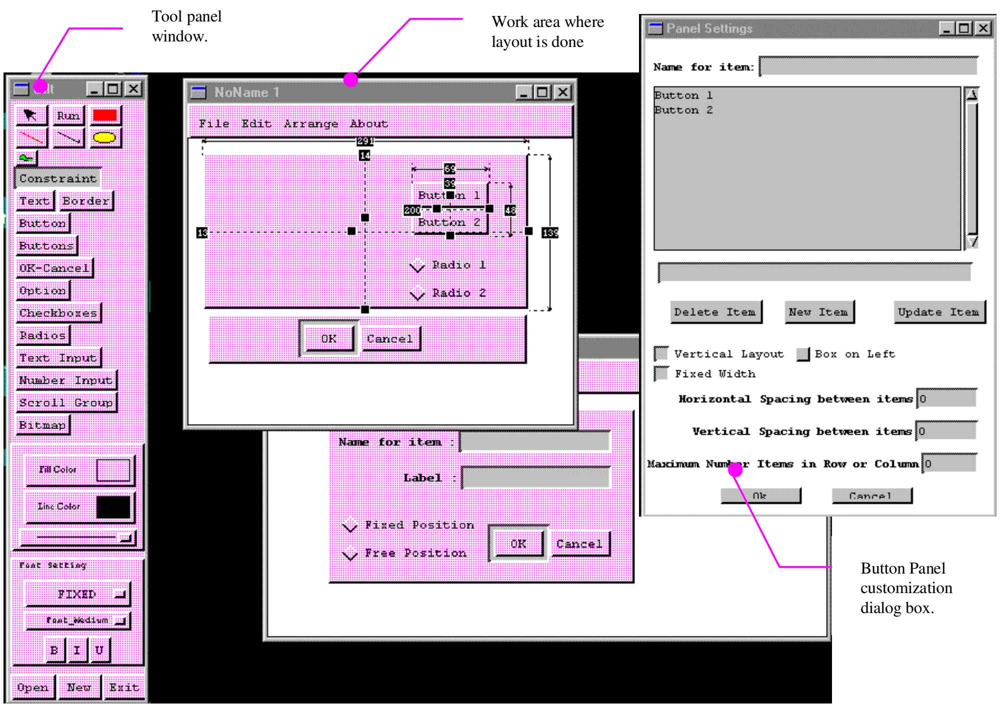

Fig 1
GILT is composed of two main components, tool panel window and work area. Tool panel contains all the predefined objects as well as constraint button. When constraint button is selected, constraint handles are enabled where as when cursor is selected selection handles are enabled. NoName1 windows shows an example of objects with constraint handles. Each object in GILT is customized with individual customization dialog box. Custom dialog box shown above, lets user to define labels of button, number of buttons, and button layout in button panel.

when value of the dependent slot changes, the slot contains constraint changes as well, but the reverse is not automatically true. Constraints are important in interface design for two reasons. First, they help to ensure an attractive interface by lining up and sizing objects correctly.
Second, they help to build more interactive interface by maintaining dynamic relationships of various properties of different objects.

GILT works like a conventional drawing program. It offers all the primitive graphical objects such as line, arc, rectangle, and so forth. GILT also provides a set of predefined widgets. New graphical objects can be created by laying out these predefined objects and grouping them together. Menubars can be created with a separate editor, which uses a collapsible tree to visualize the resulting menubar. GILT supports multiple windows. This feature allows objects to be copied from one window to another reducing excess work creating similar graphical interface and comparing different interface designs much easier. Each object has its own customization method. For example, *Customize* window in fig 1 lets user to add or delete buttons, define labels for each button, and layout of buttons in the button panel. GILT also provides thirty two different fonts, six different line styles, and twenty four different colors, which are more than enough to create a fashionable interface. A final notable feature of GILT is that it supports look and feel of all three major windowing systems, Windows, Macintosh, and X-Windows.

Related Work
There have been a number of graphical user interface development tools that allow dynamic runtime behaviors of objects to be specified and manipulated graphically. Lapidary[2, 7] allows graphical specification of constraints, but the representation of the constraints is limited to a dialog box, that is not visually related to objects being manipulated. Although some systems such as OPUS[3], represents constraints as graphical objects which can be attached to the objects to specify constraints. The drawback here, though, is that behaviors of objects can not be viewed until other separate process interprets the specifications. More interactive tools, such as Rockit[4] provide instant feedback of specified behaviors. However they often have a complicated interface to distinguish different kinds of constraints. GILT attempts to address these problems. GILT allows constraints to be specified and visualized directly on the objects and gives an instant feedback. GILT

uses simple mathematical notations and user defined function names to describe different constraints.

Euphoria[5] uses handles to identify properties of objects and allows constraints to be specified by dragging mouse from one handle to another. Each handle in Euphoria often presents multiple properties of an object. This leads to the constraints having to be set up on unnecessary properties of objects to achieve a certain behavior. Further, the placement of handles is not intuitive in a sense that the left-most middle handle defines width of an object rather than the left. Similarly, the top-most middle handle defines height rather than top of an object. GILT, on the other hand, uses handles to define *each* property of objects. Then placement of these handles corresponds to the natural geometric description of objects, akin to those of architectural drawings or geometry textbooks – see fig 2b and 2c.

Although these systems represent constraints graphically, they do not allow these representation objects to be directly manipulated to change properties of constraints. Galaxy[6] represents constraints with springs and struts, and allows them to be directly manipulated. However, this form of direct manipulation is limited to changing destination of the struts and springs by dragging their connections. GILT represents constraints and their properties with arrow lines and text objects. These arrow lines can be moved to change source and destination of constraints. In addition, they can also be cut to remove constraints, copied to make instance of existing constraints, and pasted to specify new constraints. Text objects can be directly edited to change the behavior of the constraints.

SPECIFICATION OF CONSTRAINTS

Constraints can be specified among all objects created by GILT including the window itself. Constraints are specified and manipulated by clicking or dragging by the constraint handles of objects. The *Gilt* window in fig 1 has a tool panel where user can choose predefined objects from. When a cursor, shown as a white arrow, is selected on the tool panel, selected objects in work area display selection handles. Users can change geometrical properties of the objects with these handles. However, when constraint is selected on the tool panel, the selected objects in the work area display constraint handles. Users can use constraint handles to specify constraints not only on geometrical properties but on other properties of objects as well. Users can select objects the same way independent of whether Constraint or the cursor is selected on the tool panel.

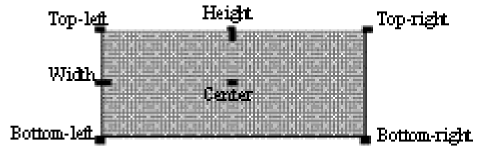

Fig 2a: The handles that Euphoria uses for both selection and constraint satisfaction. Note the positions of height and width handles. These handles are easily misinterpreted as top and left.

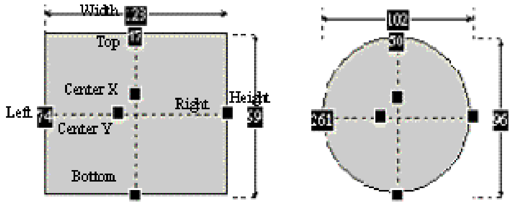

Fig 2b: Constraint widget feedback objects for objects which are defined by left, top, width and height. On the left, the handles are labeled.

Fig 2c.

A Line needs 2 points to define the coordinates because the width and height do not have enough information.

Constraint handles

Almost all of the applications of today use selection handles to alter size and position of selected objects by clicking on or dragging them. We believe that constraints should be specified in a similar manner because user interface should be consistent when user manipulates objects, whether the manipulation is to lay out the objects or to specify constraints.

Each constraint handle of GILT represents one unique property of an object. Some objects in GILT need left, top, width, and height to describe their coordinate and shape where as others need 2 explicit points. Objects of the first type have 8 handles; left, right, top, bottom, width, height, center x, and center y. Objects of the second type have 4 handles; the x and y coordinates of both the start and the end points. Layout of these handles, see fig-2a and 2b, is carefully chosen not to confuse the user while providing the user with a level of familiarity.

The clear interface is as much important as consistent interface. However, applications that use the handles to specify constraints pose some confusion to the user in defining and laying out handles. For example, Euphoria[5] uses the same set of handles to establish constraints and to select objects- see fig 2a. When a user changes properties of objects with selection handles, the user does not think about which specific property of object each handle represents. For example, when the users change height of an object with the height handle in fig 2a, they do not think about whether it represents the top or the height of the object. Only importance to users is that dragging by the handle changes height of the object. However, when users try to define a constraint, the property each handle represents becomes important for them because the constraint should be specified to specific properties of the object. The height and the width handles of Euphoria can be easily misinterpreted as the top and the left because they are placed at the top-most and left-most place of the object.

Another problem with using the same set of handles for selection and for constraint specification is that each of selection handles often represents multiple properties of an object. This can be limiting factor when one tries to define constraints. For example, if the user tries to define a constraint that would align objects at their horizontal center, it is very hard to achieve with only one center handle. Euphoria provides other means to achieve this, which is very cumbersome.
GILT employees a natural way of describing geometry of objects, that is easily found in any architectural drawings or geometry book. GILT achieves both consistent and clear interface with descriptive constraint handles. Each of these basic handles is used to define constraints that maintain a geometrical and a positional relationship among objects. Text objects are used to display values each handle represents, and constraint information when the handle is constrained. Some of the handles such as right handle do not display values because right is derived from left+width, and is not defined as a separate geometrical property.

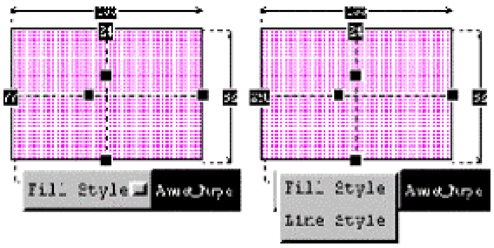

Fig 3: The optional handle which displays properties of objects which are not represented by basic points. Constraints can be specified on the black handles to the right of the option button. The handles display the values, which can be directly edited.

There is an optional handle that can be displayed from menubar of each window. This handle works like an option button with property sheet as its options - see fig 3. This handle contains a list of all the other slots of an object that are not represented by the basic handles. This optional handle allows constraints between numerical and non-numerical properties of objects. For example, user can specify a constraint that an object's position information (left, top) is displayed in text. The constraint keeps the value of text continuously updated .

Because GILT uses one-way constraints, GILT has a notion of *primary selection* and *secondary selection* where the primary selection depends on the secondary selection. Other systems use many different symbols to distinguish between constrained and unconstrained properties of objects as well as which constraints are defined. Often times these symbols are very complicated or they do not provide a strong visual information. GILT uses the color changes and the texts in the handles to indicate if handle is constrained and what kind of constraint is attached to them. Usually handles are black and display the values they represent. When a constraint is specified on a handle, it appears differently according to its roll in the constraint. A handle changes color to red if it is the primary selection, to blue if it is the secondary selection, and to purple if it is both a secondary and a primary selection. Blue and red are easily distinguishable, and catch one's

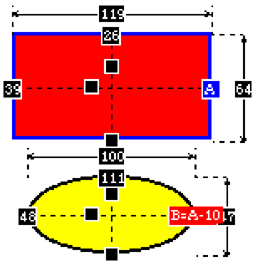

Fig 4a.
Right of the circle depends on Right of the rectangle with A relation of –10.

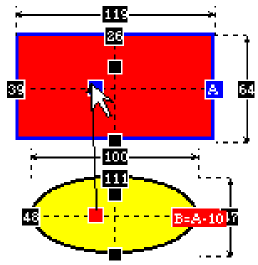

Fig4b.
A constraint is being set up by dragging the mouse from one handle to another.

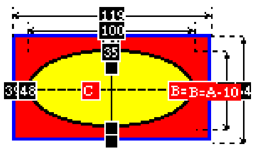

Fig 4c.
The circle instantly jumps to horizontally center itself with the rectangle.

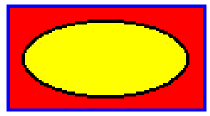

Fig 4d.
Constrained objects can be viewed with out constraint handles. If the rectangle moves

attention quickly, and mixing blue and red gives purple, that implies both the characteristics of red and blue.

A constraints on the graphics handles are about always defined as a simple equation. This equation is a form of A*1.0+0 where A can be any variable between A-Z, and 1.0 is percentage which can be any floating value, and 0 is an offset which can be any integer. Default value of the offset is 0, and the default value of the percentage for constraints whose secondary selection is either of Center_X or Center_Y handle is 0.5 and 1.0 for all other handles. This equation is displayed in the primary handle and the variable used in the equation is displayed in the secondary handle. The equation identifies the dependency of the constraint because the value of an equation depends on the value of the variable it uses. In fig 4a, the right of the rectangle is secondary selection and right of the circle is primary selection. Therefore the right handle of the rectangle is blue and displays variable 'A'. The right handle of the circle is red and displays B=A – 10, which is an equation with default percentage value omitted. From this, one can easily see that the right of the circle depends on the right of the rectangle with the relation of right -10.

Two types of constraints can be specified on the optional handle. When the primary and secondary handles contain the same type of values, an equal constraint is specified, which displays "=" in handle, and if not, a translation constraint is set up. This constraint translates values from one type to another with provided inferences from user as described in Spaghetti Paper[8].

METHODS OF CONSTRAINT SPECIFICATION
There are four ways to define and modify constraints.

- Set up constraints by graphically dragging between primary and secondary handles.
- Set up constraints by direct editing the text in handles.
- Set up constraints by copying and pasting existing constraints.
- Use dialog box to modify constraints.

Graphically connecting handles together
Primary and secondary handle selection uses only left mouse button. GILT can not use the right or middle mouse button because GILT is designed for all three platforms that Amulet supports. The handle on which mouse button is pressed is selected as a primary handle. While the mouse is dragged around, two feedback events happen.

1. An arrow rubber band line points from the primary handle to current cursor position. Direction of the rubber band line is opposite of the data flow. This is because the constraints are of the form "my value is computed from that object's value". This is analogous to the spreadsheet formula. Therefore, it seemed that the arrow should point toward the object that is depended on. Note that this is opposite from the direction of the data flow. Our initial user surveys suggest an even split on user preferences for which way the arrow should point. Another area of contention is which object should move when two objects are connected by dragging. We move the initial object because arrow points from the initial object to the current cursor position. It is more natural for an object to move from starting point of an arrow to the head of an arrow than the opposite. We will continue to evaluate this issue as we get more user experiences.
2. The color of the primary handle changes to red indicating that the new constraint is being created on the handle. In addition, whenever the mouse cursor is over any of the handles, color of the handle changes to blue indicating that the handle is being

selected as secondary handle. After the constraint is setup, color of the primary and secondary handle change to what they suppose to be according to their state.

Specifying the constraint this way on basic handles yields an equal constraint with default values of offset and percentage. As soon as dragging is done, objects jump to new position or resize immediately. Attempts to change the constrained properties of objects returns a warning that allows user to choose from various options.

Direct text manipulation.

GILT lets user to directly edit the text in handles to specify, remove, and change the properties of constraints. Four different equation forms are accepted by basic (geometry) handles.

1. Constant
2. $ * P
3. $ + O
4. $ * P + O

where $ stands for any variables between A – Z, O stands for offset that can be any integer, and P stands for percentage that can be any floating value.

The constant value accepted by handles, allow dynamic changes of the properties of objects without specifying constraints. User can specify a new constraint with equations 2 – 4 where $ is a variable assigned to the dependent handle. Existing constraints are manipulated by changing offset, percentage, and variable in the equation. Changing variable in the equation causes the object to depend on another object. These techniques can be used for constraints specified by method from previous section. Also, constraints set up with text editing can be modified and removed with the dialog box.

Text display in handle might crowd the work area. Text display can be turned off from the main menu of each window. However, color display is not turned off to provide minimum visual presentation of constraints.

Deriving constraints from existing one using visualization objects

One of the important innovations in GILT's constraints is that the constraint should be visualized and treated as a graphical object. Although some other systems allow constraints to be visualized graphically, they do not allow these visualization objects to be graphically manipulated. Another issue of constraint visualization is that often times, visualization of the constraint can not be removed unless the constraint is removed. As more constraints are defined, the more crowd the work area gets. Some of the systems such as OPUS allow interface to be broken into sections to reduce number of visualizations of constraint displayed at once. However, this does not help to view the interface as a whole and the constraints among individual objects.

In addition to the handles and the text labels, gilt can also display another visualization objects. These objects are composed of an arrow line and a text object. The arrow represents constraint dependency, and the text displays constraint information like text in handles. GILT usually hides these objects, and displays them upon request from the user. The *Show Constraints* command in menu displays all the constraints attached to the selected objects, and Ctrl+Left_Click on an individual handle displays constraints attached to that handle. In figure 5b, GILT forces objects related to the button panel to be selected to show the full relationships. The texts on the arrow lines display equations described in previous section. However, note that variable in the equation is generalized to '$' rather than specific variable between A-Z. This is because, they describe property of constraints whereas text in handles define property of the object induced by constraints. This is important distinction from other systems because other systems do not visualize constraint itself but only behaviors of different value of different objects. GILT can treat constraints as graphical objects that can be manipulated like any other graphical objects because it can visualize constraints independent of context. From here on, we refer to the constraint visualization object described in this section as a constraint object.

The constraint objects can be selected using drag in background and by single clicking on an arrow. Feedback for selected the constraint

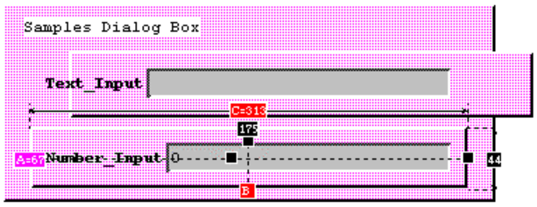

Fig 5a.
One object, number input widget, is selected. This widget has constraints on left, width, and bottom handle.

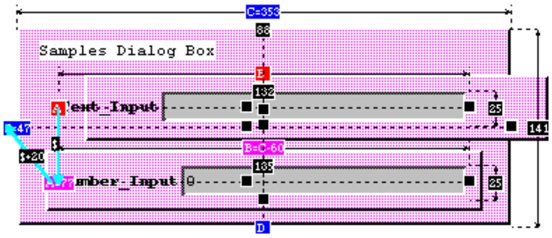

Fig 5b.
When left handle of the widget is clicked, constraints attached to the handle are displayed. This forces other objects selected.

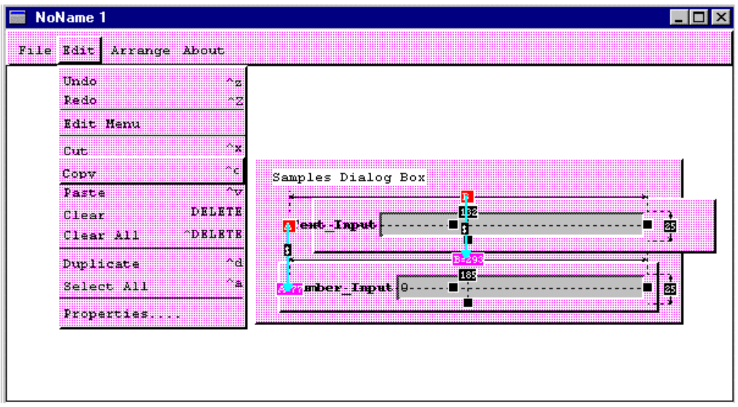

Fig 6a.
Constraints attached on left and top handles of text input widget are copied using the commands in the manubar.

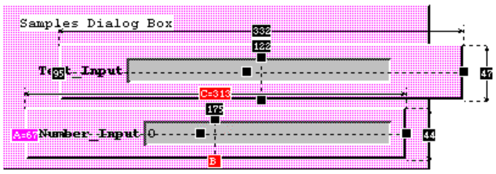

Fig 6b.
Two border rectangles are selected.

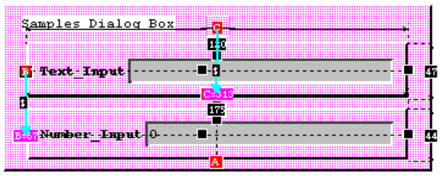

Fig 6c.
Using Paste command from the menu, user can automatically specify constraints for the selected rectangles.

object has two handles one at each end. Selected constraint objects can be cut, copied, and pasted with regular commands from the edit menu. These operation are done to constraint itself as well as the constraint object it is visualized by. Cut removes constraint, Copy makes a copy of selected constraint, and Paste visualizes the copied constraint in work space little bit left to the original one just like any other object manipulation.

Property of the constraints can be modified by directly editing the text of constraint object, in the same way as described previously. Once constraint object is pasted and the constraint information is modified, the user can specify primary and secondary handle for this constraint. The user can drag the handle at either end of the arrow and drop it on any of the handles. A rubber band line moves with the mouse while dragging. When dragging ends on a handle, the constraint object changes by attaching dragged end to the handle. The arrow tip selects primary handle and the other end selects the secondary handle when they are dropped in a handle. User can select different handles in same manner after they are dropped. User can move around an entire arrow by the middle of the arrow. When entire arrow is moved, actual arrow moves around instead of fake feedback object. While arrow is moving around, it might be dropped on a handle by accident. GILT disregards the drop if arrow is moving around by the middle of an arrow rather than either ends. After both ends are dropped, objects move or resize in respond to the new constraint specified.

GILT generalizes selected constraints between two objects. For example, in fig 6a-c, two constraints are copied from the button panel and the radio button panel. These constraints are pasted to the selected text input and integer input widgets. These pasted constraints find their primary and secondary handles automatically and specify new constraints between these two objects. This idea is further generalized. GILT will eventually generalize constraints among more than two objects. Copied constraints can then be pasted to different set of objects to create same behavior among the objects. Another important use of copy and paste comes from multi-constraint. Amulet allows a value of an object to depend on multiple values by allowing multiple constraints to be stored in one slot. Users can build a multi-constraint by copy-paste existing constraints and then bond them together at the end. Multi-constraints could also be built from copying existing multi-constraint and deleting or adding more constraints to it. Multi-constraints created this way can be generalized so that when it is pasted to a handle, individual constraints can find their destinations automatically.

Manipulation through a dialog box

Constraints can be manipulated or removed with a dialog box. The dialog box is brought up by double clicking on any of the handles or any of the constraint objects – see fig 7. The user has 4 options to choose from. The user can change property of constraint with new values, remove the existing constraint, define custom constraint, and cancel. When a constraint is remove, visualization objects change according to their state. Eventually the Custom option will let users to define their own constraints with scripting language provided by GILT. These constraints are generalized to work with any run-time object. If the user changes the properties, objects are immediately affected by the changes.

Current status and Future Work

Gilt is being developed as a part of the Amulet project. Its design is completed, and significant portion of the implementation is complete. Current limitation of GILT is that it has limited set of predefined constraints. More sophisticated constraint are needed to build more dynamic and interactive interface. However, the set of constraints GILT currently offers is sufficient enough for any interface layout needs.

The main problem we will look for is to define high-level constraints without the user's writing any code. In current design, the user has to write custom constraints. We will investigate other ways to define constraint such that the definition process does not break continuous interaction.

GILT will be included in version 4 of the Amulet, which will be released in coming April.

References

1. Brad A. Myers. The Amulet Environment: New Models for Effective User Interface Software Development. CMU-CS-96-189 and Human Computer Interaction Institute Technical Report CMU-HCII-96-104

2. Brad A. Myers. Creating Graphical Interactive Application Objects by Demonstration. *Proceedings* of the ACM SIGGRAPH (Williamsburg, VA, Nov, 1989), ACM Press, 95 – 104.

3. Scott E. Hudson. Interactive Specification of Flexible User Interface Display. ACM Transactions on Information Systems, Vol. 8, No. 3, July 1990, p269-288

4. Solange Karsenty. Inferring Graphical Constraints with Rockit. Proceedings of HCI'92, University of York, U.K., September 1992.

5. T. Paul McCartney. EUPHORIA Reference Manual. Washington University technical report WUCS-95-19

6. Galaxy Reference Manual, p50-61.

7. Brad Vander Zanden, Demonstrational and Constraint-Based Techniques for Pictorially Specifying Application Objects and Behaviors. ACM Transactions on Cmputer-Human Interaction, Vol. 2, No. 4, December 1995, p 308-356.

8. Brad A. Myers. Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs. ACM SIGGRAPH Symposium on User Interface Software and Technology. Hilton Head, SC, 1991, p 211-220.