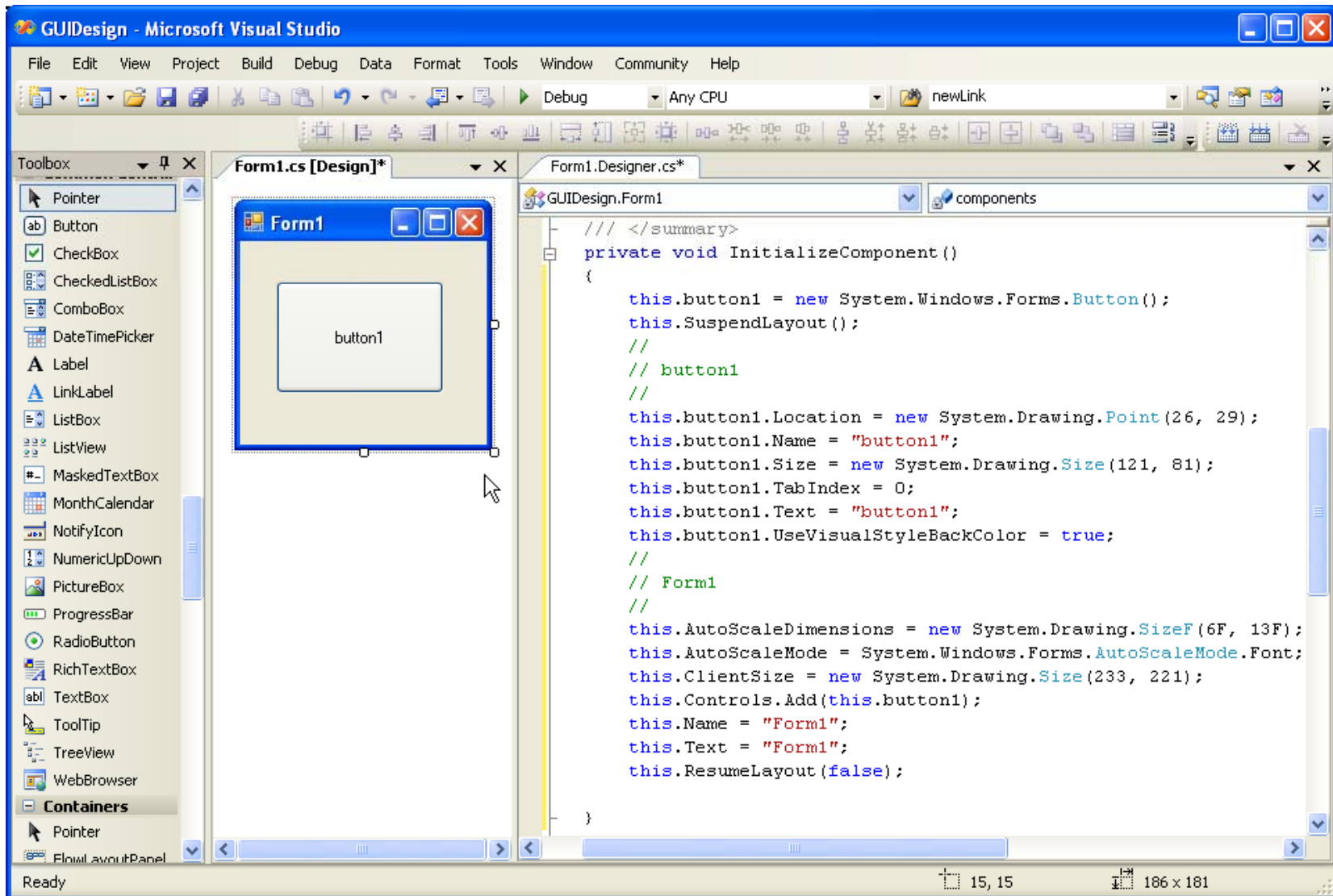




The Auckland Interface Model (AIM)

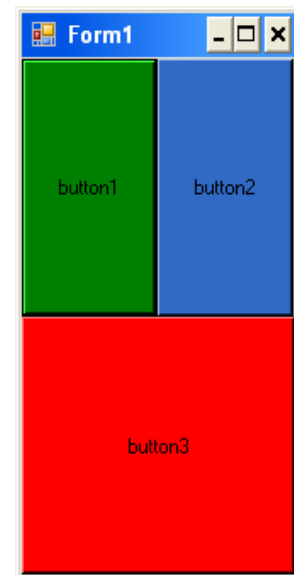
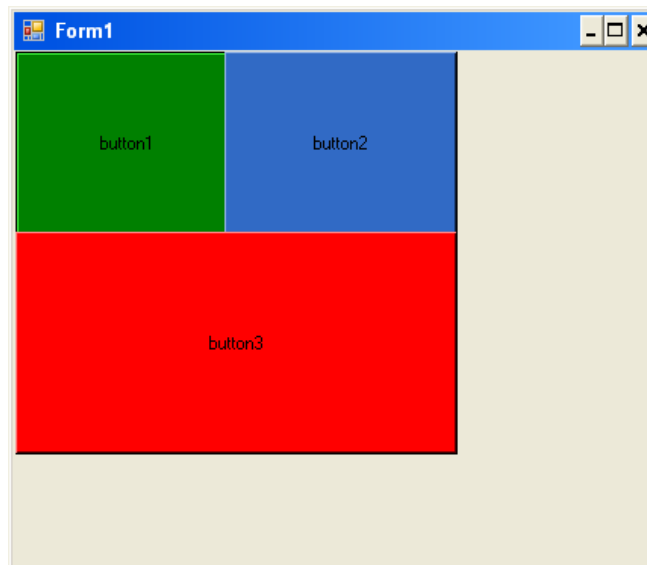
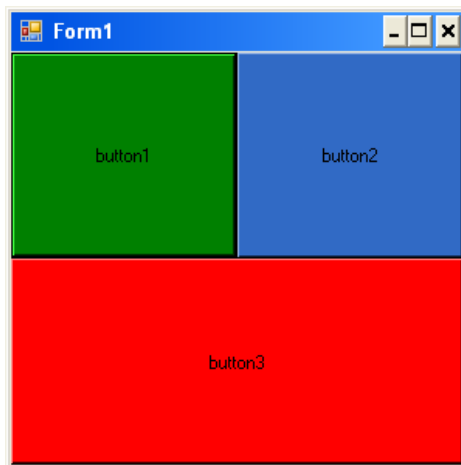
HCI Guest Lecture
Christof Lutteroth

Developing GUIs

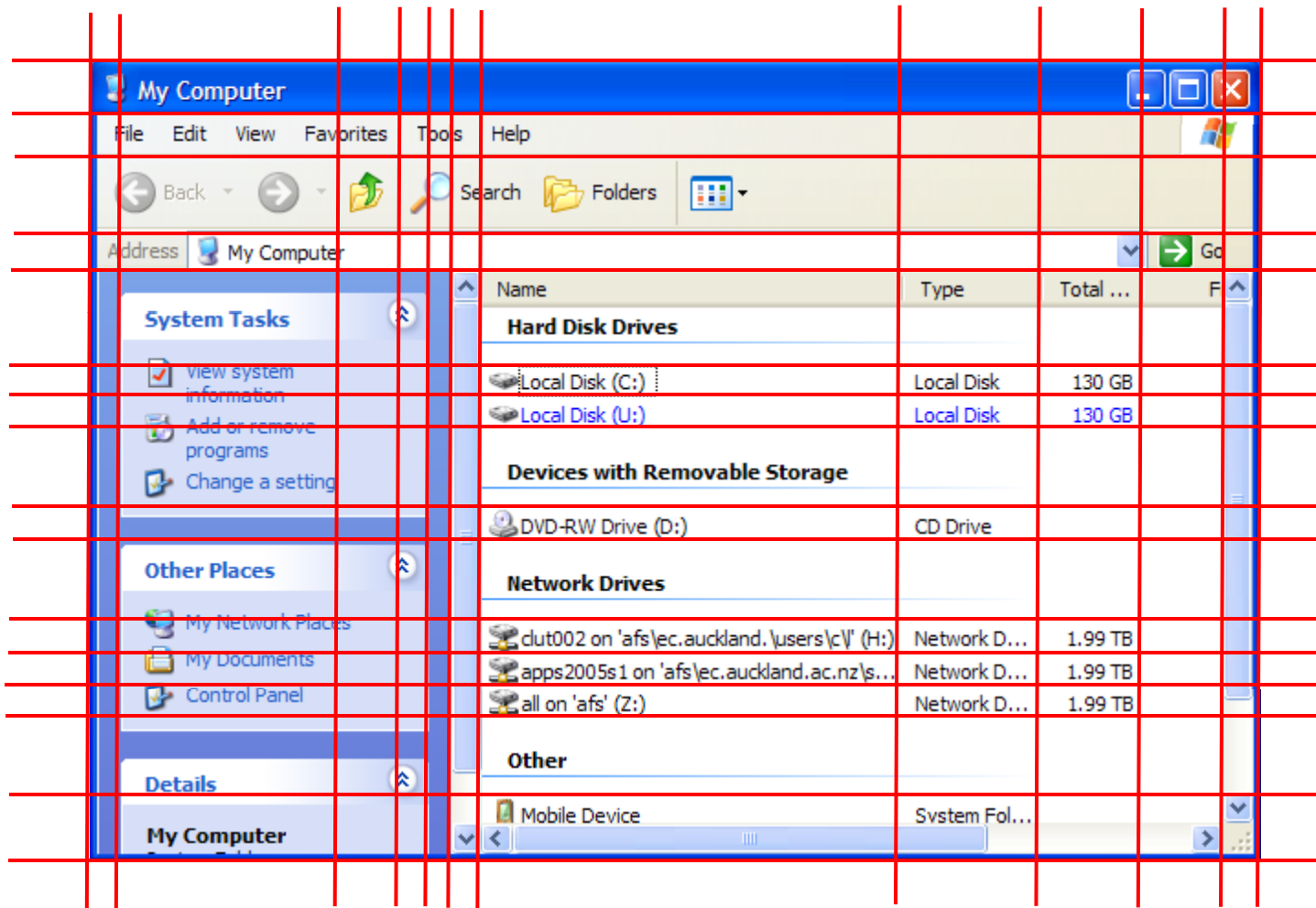


Layout Managers

1. Get a layout specification as input
2. Recalculate the positions and sizes of the controls after each resizing

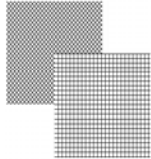


Specifying GUI Layout



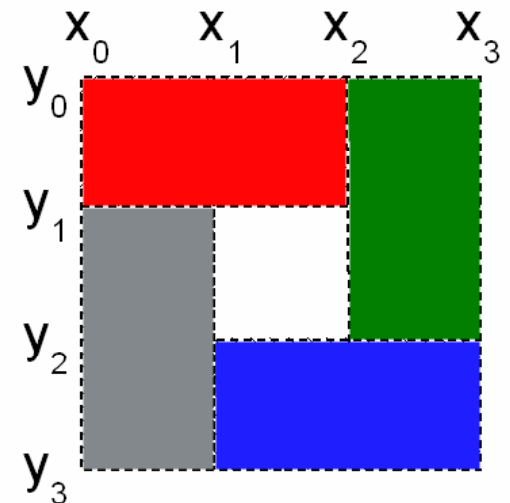
Controls are aligned in a grid

Areas

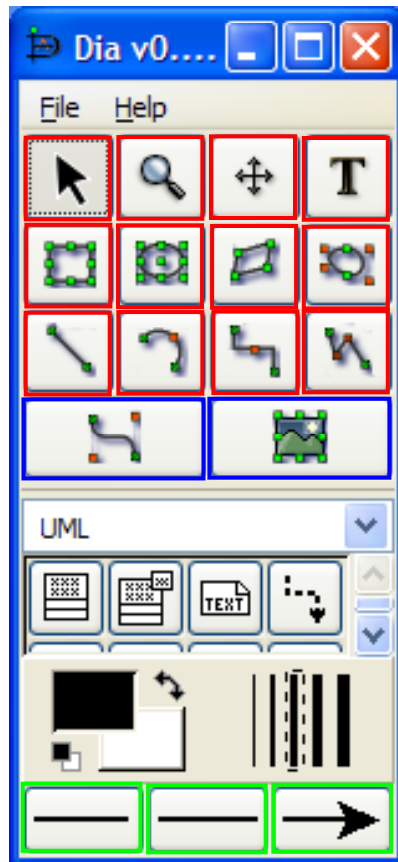
- Grid lines are variables with coordinates (tabs)
- Place controls by choosing left, top, right and bottom tab (area)
- Overlapping areas with layers 

$a =_{def} (x_1, y_1, x_2, y_2, layer, content)$

$A = \{(x_0, y_0, x_2, y_1, 0, red), (x_2, y_0, x_3, y_2, 0, green),$
 $(x_1, y_2, x_3, y_3, 0, blue), (x_0, y_1, x_1, y_3, 0, grey),$
 $(x_1, y_1, x_2, y_2, 0, empty)\}$



Specifying GUI Layout



Same size

Same size

Same height as above

Double width as above

Same size

2/3 width as above

Linear Constraints

$$C \subset \left\{ \begin{array}{l} a_0x_0 + \dots + a_mx_m + b_0y_0 + \dots + b_ny_n \text{ OP } c \\ | \quad a_0, \dots, a_m, b_0, \dots, b_n, c \in \mathbb{R} \wedge \text{OP} \in \{\leq, =, \geq\} \end{array} \right\}$$

■ Absolute constraints

$$x_3 = 50. \quad x_2 - x_1 = 100.$$

■ Relative constraints

□ Relative position

$$x_2 - x_1 = x_3 - x_2 \Leftrightarrow -x_1 + 2x_2 - x_3 = 0. \quad x_1 \text{---} x_2 \text{---} x_3$$

□ Relative size

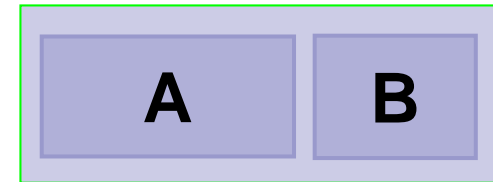
$$x_2 - x_1 = 2(x_4 - x_3) \Leftrightarrow -x_1 + x_2 + 2x_3 - 2x_4 = 0.$$

□ Aspect ratio

$$\frac{x_2 - x_1}{y_2 - y_1} = \frac{16}{9} \Leftrightarrow -x_1 + x_2 + \frac{16}{9}y_1 - \frac{16}{9}y_2 = 0.$$

■ Different units possible per constraint (cm, pixels)

Problem



Designers working on a GUI:

- “Button A should be twice as wide as button B” ($width_A = 2 width_B$)
- “Button B should be 100 pixels wide”
- “The GUI should be no more than 200 pixels wide”

Infeasible!



Linear Programming

Input:

- Set of linear **constraints** C on variables

$$C \subset \left\{ \begin{array}{l} a_0x_0 + \dots + a_mx_m + b_0y_0 + \dots + b_ny_n \text{ OP } c \\ | \quad a_0, \dots, a_m, b_0, \dots, b_n, c \in \mathbb{R} \wedge \text{OP} \in \{\leq, =, \geq\} \end{array} \right\}$$

- Linear **objective function** to minimize

$$c_1x_1 + c_2x_2 + \dots + c_nx_n$$

Output: **variable values** so that...

- All constraints are satisfied
- The value of the objective function is minimal



Soft Constraints

$width_A = 2 width_B$ is a **hard** constraint

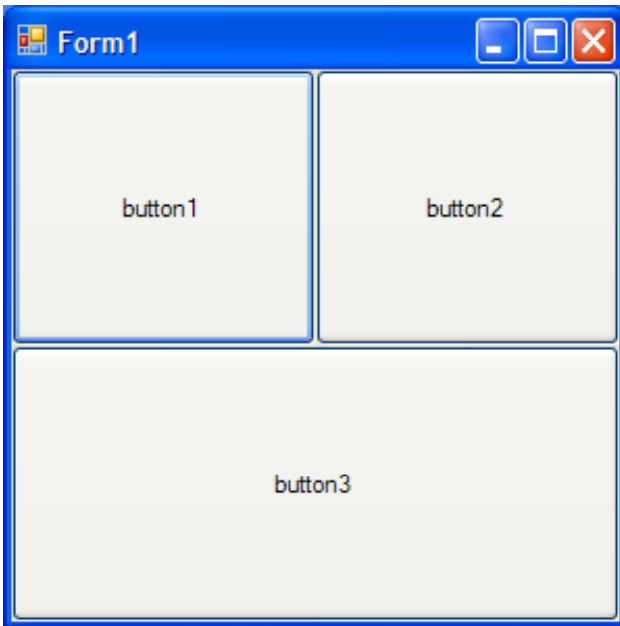
1. Allow $width_A$ to be smaller or bigger

$$width_A + d_1 - d_2 = 2 width_B \quad d_1 \geq 0, d_2 \geq 0$$

2. $width_A$ should not be much smaller/bigger

$$objective\ function = d_1 + d_2$$

Example 1



```
LayoutSpec ls = new LayoutSpec();  
XTab x1 = ls.AddXTab();  
YTab y1 = ls.AddYTab();
```

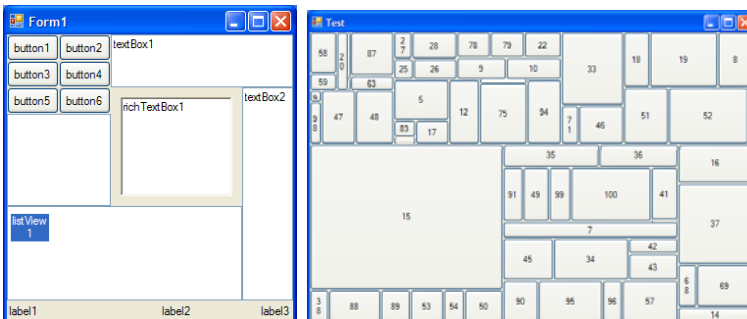
```
ls.AddArea(ls.Left, ls.Top, x1, y1,  
button1);
```

```
ls.AddArea(x1, ls.Top, ls.Right, y1,  
button2);
```

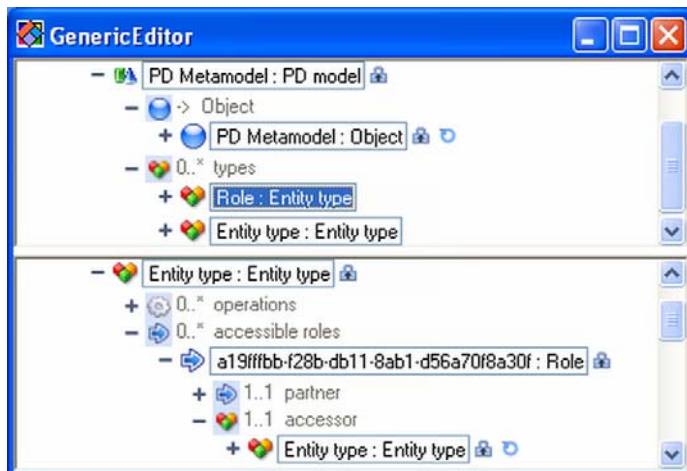
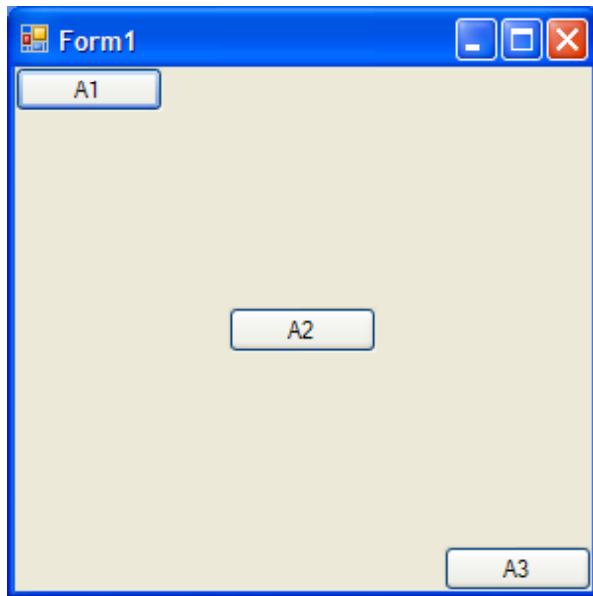
```
ls.AddArea(ls.Left, y1, ls.Right, ls.Bottom,  
button3);
```

```
ls.AddConstraint(new double[] { 2, -1 },  
new Variable[] { x1, ls.Right },  
OperatorType.EQ, 0);
```

```
ls.AddConstraint(new double[] { 2, -1 },  
new Variable[] { y1, ls.Bottom },  
OperatorType.EQ, 0);
```



Example 2



```
LayoutSpec ls = new LayoutSpec();  
Column c1 = ls.AddColumn();  
Row r1 = ls.AddRow();  
Row r3 = ls.AddRow();  
r1.Next = r3;  
Row r2 = ls.AddRow();  
r2.InsertAfter(r1);  
// ...
```

```
Area a1 = ls.AddArea(r1, c1, b1);  
a1.HAlignment =  
    HorizontalAlignment.LEFT;  
a1.VAlignment =  
    VerticalAlignment.TOP;  
Area a3 = ls.AddArea(r3, c1, b3);
```

```
r2.HasSameHeightAs(r1);  
r3.HasSameHeightAs(r1);
```



Conclusion

- We can specify GUI layout with:
 - Tabstops
 - Areas
 - Linear constraints
- Linear programming makes it possible to transform hard constraints into soft constraints
- Link to project website on my homepage:
<http://www.cs.auckland.ac.nz/~lutteroth/>

Thank you for your attention ☺



AIMM

OTHER AIM PROJECTS

AIM Editor

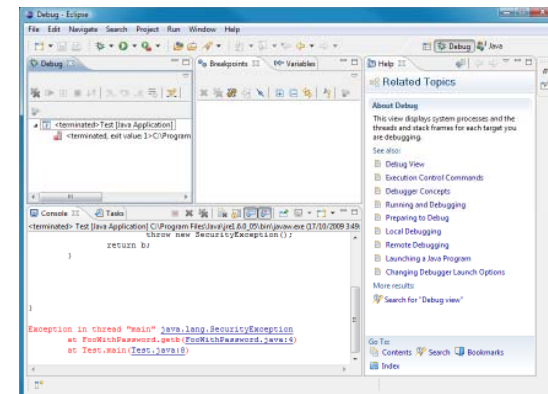
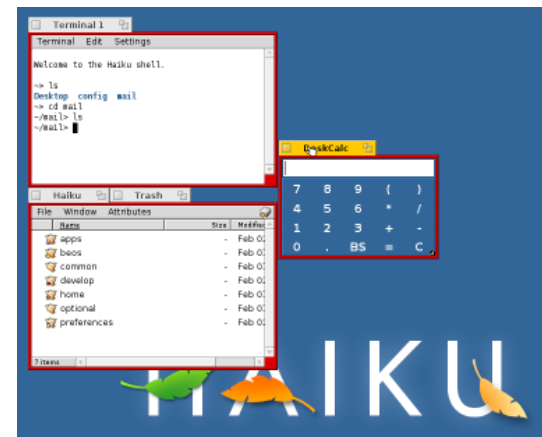
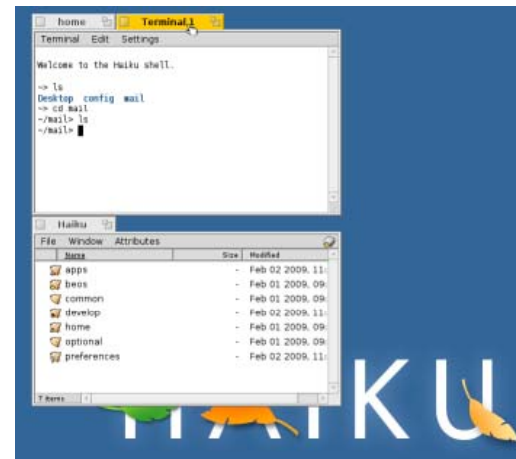
- AIM comes with a built-in GUI layout editor
 - Call `AIMLayout.Edit()` and you can change the current GUI
 - Drag&drop of areas and visual specification of constraints
 - Can also load/save the layout
-
- Work in progress
 - Make it a complete GUI editor
 - Support for adding completely new controls, more visual constraint editing, ...



<http://genoupe.se.auckland.ac.nz/aimf>

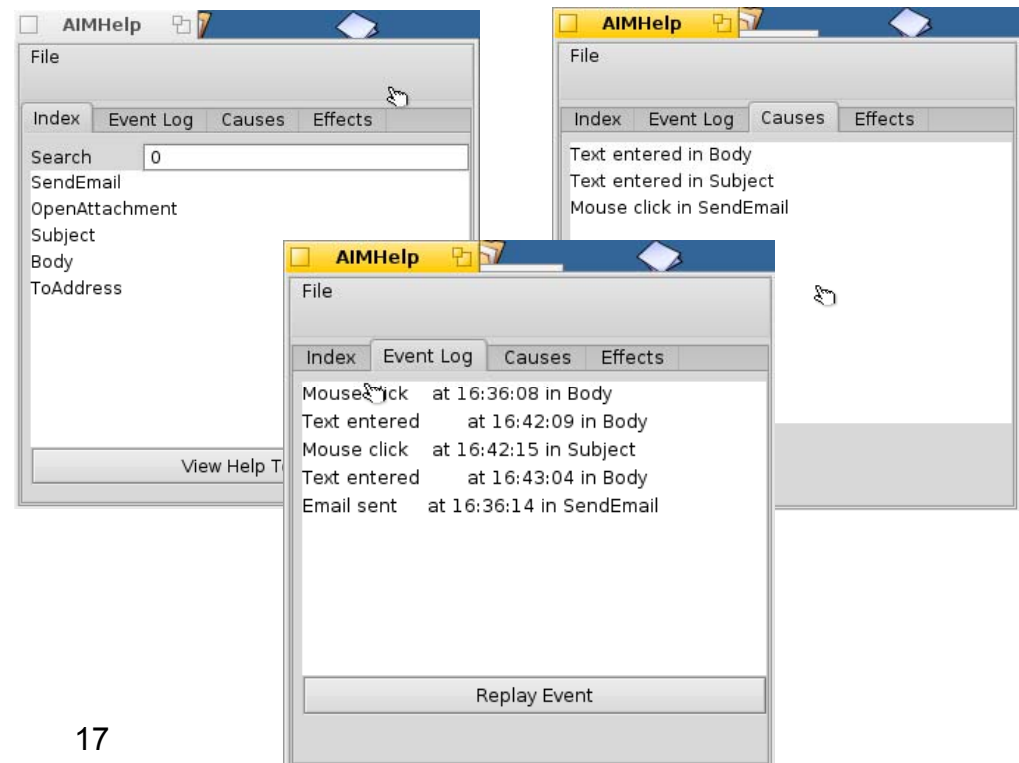
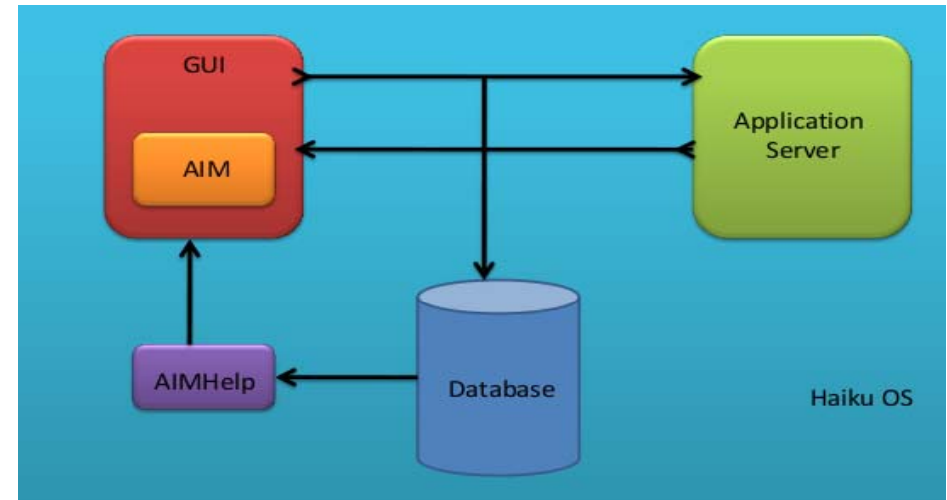
Stack & Tile

- New feature for a windowing system (in Haiku OS)
- Hold Windows key down to Stack&Tile windows
 - Stack window onto another by overlapping their tabs
 - Tile window to other windows by bringing edges close together
- Stack&Tile configuration is persistent (if supported by app)



HaikuHelp

- Generate help information automatically by analyzing input messages and state changes in a GUI
- Features:
 - Automated Help Index
 - Event Log
 - Causes Analysis
 - Effects Analysis





Thank You for Your Attention !

Questions and comments?

→ lutteroth@cs.auckland.ac.nz