Avoiding Faulty User Interfaces

Jaakko Järvi

Feb 11th, 2014
- UIs are perhaps the most costly area of software
- Observations in a major desktop software company:
  - 30+% of all code is in UI logic
  - 60+% of all defects in UI code
UIs are perhaps the most costly area of software

Observations in a major desktop software company:
- 30+% of all code is in UI logic
- 60+% of all defects in UI code

GUI programming is not getting easier:
- Multitude of platforms, devices, screen sizes, etc. to support
- Responsiveness harder (latencies, failures in updating UI state)
UIs are perhaps the most costly area of software

Observations in a major desktop software company:
- 30+% of all code is in UI logic
- 60+% of all defects in UI code

GUI programming is not getting easier:
- Multitude of platforms, devices, screen sizes, etc. to support
- Responsiveness harder (latencies, failures in updating UI state)

The difficulty of UI programming underestimated
This is all too common
This is all too common

why?
def ChangeCurrentHeightPx(self, event):
    self.LastUpdated = "Height"
    constrained = self.Controls["Constrain"].GetValue()
    # no matter what the percent & current stay bound together
    # get current height, and compute relative height and place new rel. ht
    height = float(self.Controls["AbsolutePx"]['"Height"'].GetValue())
    pct = height / self.InitialSize[self.Height]
    self.Controls["Relative%"]['"Height"'].SetValue(str(pct*100))
    if constrained: # update width & width%
        width = self.Controls['AbsolutePx'] ['"Width"'].GetValue(str(pct*100))
        self.Controls['AbsolutePx'] ['"Height"'].SetValue(str(round(height)))

def ChangeCurrentHeightPct(self, event):
    self.LastUpdated = "Height"
    constrained = self.Controls["Constrain"].GetValue()
    # no matter what the percent & current stay bound together
    # get current rel. ht, and compute absolute height and place new abs. ht
    height = float(self.Controls["Relative%"] ['"Height"'].GetValue())
    cur = height * self.InitialSize[self.Height] / 100
    self.Controls['AbsolutePx']['"Height"'].SetValue(str(round(cur)))
    if constrained: # update width & width%
        width = float(self.Controls['Relative%'] ['"Width"'].GetValue())
        self.Controls['Relative%']['"Height"'].SetValue(str(width))
        height = width * self.InitialSize[self.Height] / 100
        self.Controls['AbsolutePx']['"Height"'].SetValue(str(round(height)))

def ChangeConstrainState(self, event):
    constrained = self.Controls["Constrain"].GetValue()
    # If the ratio is constrained, determine which dimension
    # was last updated and update the OTHER dimension.
    # For example: if Height was last updated, use Height as
    # Width's new percent, and update Width's absolute value
    if constrained:
        if self.LastUpdated == "Height":
            pct = float(self.Controls["Relative%"]['"Height"'].GetValue())
            self.Controls["Relative%"]['"Width"'].SetValue(str(pct))
            width = pct * self.InitialSize[self.Width] / 100
            self.Controls['AbsolutePx']['"Width"'].SetValue(str(round(width)))
        else:
            pct = float(self.Controls["Relative%"]['"Width"'].GetValue())
            self.Controls["Relative%"]['"Height"'].SetValue(str(pct))
            height = pct * self.InitialSize[self.Height] / 100
            self.Controls['AbsolutePx']['"Height"'].SetValue(str(round(height)))
    else:
        # no matter what the percent & current stay bound together
        # get current rel. wd, and compute absolute width and place new abs. wd
        width = float(self.Controls['Relative%'] ['"Width"'].GetValue())
        cur = width * self.InitialSize[self.Width] / 100
        self.Controls['AbsolutePx']['"Width"'].SetValue(str(round(cur)))
        if constrained: # update height & height%
            height = width * self.InitialSize[self.Height] / 100
            self.Controls['AbsolutePx']['"Height"'].SetValue(str(round(height)))
            self.Controls["Relative%"]['"Height"'].SetValue(str(height))

# If the ratio is constrained, determine which dimension
# was last updated and update the OTHER dimension.
# For example: if Height was last updated, use Height as
# Width's new percent, and update Width's absolute value
# if constrained:
#     if self.LastUpdated == "Height":
#         pct = float(self.Controls["Relative%"]['"Height"'].GetValue())
#         self.Controls["Relative%"]['"Width"'].SetValue(str(pct))
#         width = pct * self.InitialSize[self.Width] / 100
#         self.Controls['AbsolutePx']['"Width"'].SetValue(str(round(width)))
#     else:
#         pct = float(self.Controls["Relative%"]['"Width"'].GetValue())
#         self.Controls["Relative%"]['"Height"'].SetValue(str(pct))
#         height = pct * self.InitialSize[self.Height] / 100
#         self.Controls['AbsolutePx']['"Height"'].SetValue(str(round(height)))

Jaakko Järvi
Avoiding Faulty User Interfaces
Feb 11th, 2014
GUI programming from the point of view of the developer

Typical event handling code (implementing most basic functionality)

def ChangeCurrentHeightPx(self, event):
    self.LastUpdated = "Height"
    constrained = self.Controls["Constrain"].GetValue()
    if constrained:
        # no matter what the percent & current stay bound together
        # get current height, and compute relative height and place new rel. ht
        height = float(self.Controls["AbsolutePx"]["Height"].GetValue())
        pct = height / self.InitialSize[self.Height]
        self.Controls["Relative%"]["Height"].SetValue(str(pct*100))
        width = self.Controls["AbsolutePx"]["Width"].GetValue()
        if width:
            self.Controls["Relative%"]["Width"].SetValue(str((pct * width) / 100))
        else:
            self.Controls["Relative%"]["Width"].SetValue("-1")
    else:
        # update width & width%
        self.Controls["Relative%"]["Width"].SetValue(str(pct*100))
        width = pct * self.InitialSize[self.Width]
        self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeCurrentHeightPct(self, event):
    self.LastUpdated = "Height"
    constrained = self.Controls["Constrain"].GetValue()
    if constrained:
        # no matter what the percent & current stay bound together
        # get current rel. ht, and compute absolute height and place new abs. ht
        height = float(self.Controls["Relative%"]["Height"].GetValue())
        cur = height * self.InitialSize[self.Height] / 100
        self.Controls["AbsolutePx"]["Height"].SetValue(str(round(cur)))
        width = self.Controls["AbsolutePx"]["Width"].GetValue()
        if width:
            self.Controls["Relative%"]["Width"].SetValue(str(height))
            width = height * self.InitialSize[self.Width] / 100
            self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
        else:
            self.Controls["Relative%"]["Width"].SetValue("-1")
    else:
        # update width & width%
        self.Controls["Relative%"]["Width"].SetValue(str(pct*100))
        width = pct * self.InitialSize[self.Width]
        self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

def ChangeConstrainState(self, event):
    constrained = self.Controls["Constrain"].GetValue()
    if constrained:
        if self.LastUpdated == "Height": # update width px & %
            pct = float(self.Controls["Relative%"]["Height"].GetValue())
            self.Controls["Relative%"]["Width"].SetValue(str(pct))
            width = pct * self.InitialSize[self.Width] / 100
            self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
        else: # update width px & %
            pct = float(self.Controls["Relative%"]["Width"].GetValue())
            self.Controls["Relative%"]["Height"].SetValue(str(pct))
            height = pct * self.InitialSize[self.Height] / 100
            self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))
    else:
        if self.LastUpdated == "Height": # update width px & %
            pct = float(self.Controls["Relative%"]["Height"].GetValue())
            self.Controls["Relative%"]["Width"].SetValue(str(pct))
            width = pct * self.InitialSize[self.Width] / 100
            self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
        else: # update width px & %
            pct = float(self.Controls["Relative%"]["Width"].GetValue())
            self.Controls["Relative%"]["Height"].SetValue(str(pct))
            height = pct * self.InitialSize[self.Height] / 100
            self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))
GUI programming from the point of view of the developer

Incidental algorithm

def ChangeHeightPx(self, event):
    constrained = self.Controls["Constrain"].GetValue()
    # no matter what the percent & current stay bound together
    # get current height, and compute relative height and place new rel. ht
    height = float(self.Controls["AbsolutePx"]["Height"]).GetValue() / self.InitialSize["Height"]
    pct = height / self.InitialSize["Height"]
    self.Controls["Relative%"]["Height"]["Value"].SetValue(str(pct*100))
    self.Controls["AbsolutePx"]["Height"]["Value"].SetValue(str(round(height)))

    def ChangeHeightPct(self, event):
        constrained = self.Controls["Constrain"]["Value"]
        # no matter what the percent & current stay bound together
        # get current rel. ht, and compute absolute height and place new abs. ht
        height = float(self.Controls["Relative%"]["Height"]["Value"]).GetValue() / self.InitialSize["Height"]
        self.Controls["AbsolutePx"]["Height"]["Value"].SetValue(str(round(height)))

    def ChangeConstrainState(self, event):
        constrained = self.Controls["Constrain"]["Value"]
        # if the ratio is constrained, determine which dimension was last updated and update the OTHER dimension.
        if constrained:
            if self.LastUpdated == "Height":
                # update width px & %
                width = float(self.Controls["Relative%"]["Width"]["Value"]) / self.InitialSize["Width"]
                self.Controls["AbsolutePx"]["Height"]["Value"].SetValue(str(round(width)))
                self.Controls["Relative%"]["Height"]["Value"].SetValue(str(width))
            else:
                # update height px & %
                height = float(self.Controls["Relative%"]["Height"]["Value"]) / self.InitialSize["Height"]
                self.Controls["AbsolutePx"]["Width"]["Value"].SetValue(str(round(height)))
                self.Controls["Relative%"]["Width"]["Value"].SetValue(str(height))
Observation 1

Programming user interfaces constitutes a significant portion of all programming effort.
GUI programming from the point of view of the user
GUI programming from the point of view of the user

Poor quality GUIs are frustrating!
A concrete instance of GUI frustration

### Activity 1

**Organization / Activity**
SPEECH CLUB

**Description**
EXTEMPORANEOUS SPEAKING

**Activity 1 level**
LOCAL

**Participation Details for Activity 1**
(Use whole numbers only, no fractions.)

<table>
<thead>
<tr>
<th>Year</th>
<th>Position(s) Held</th>
<th>Were You Elected?</th>
<th>Hours/week</th>
<th>Weeks/year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh</td>
<td>MEMBER</td>
<td>No</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>Soph</td>
<td>MEMBER</td>
<td>No</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>Junior</td>
<td>SECRETARY</td>
<td>Yes</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>Senior</td>
<td>PRESIDENT</td>
<td>Yes</td>
<td>20</td>
<td>18</td>
</tr>
</tbody>
</table>

### Activity 2

**Organization / Activity**
CHESS CLUB

**Description**
CHESS

**Activity 2 level**
LOCAL

**Participation Details for Activity 2**
(Use whole numbers only, no fractions.)

<table>
<thead>
<tr>
<th>Year</th>
<th>Position(s) Held</th>
<th>Were You Elected?</th>
<th>Hours/week</th>
<th>Weeks/year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Soph</td>
<td>MEMBER</td>
<td>No</td>
<td>10</td>
<td>18</td>
</tr>
</tbody>
</table>
**A concrete instance of GUI frustration**

### Activity 1

<table>
<thead>
<tr>
<th>Organization / Activity</th>
<th>Activity 1 level</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEECH CLUB</td>
<td>LOCAL</td>
</tr>
</tbody>
</table>

**Description**

*EXTEMPORANEOUS SPEAKING*

**Participation Details for Activity 1 (Use whole numbers only, no fractions.)**

<table>
<thead>
<tr>
<th>Year</th>
<th>Position(s) Held</th>
<th>Were You Elected?</th>
<th>Hours/week</th>
<th>Weeks/year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh</td>
<td>MEMBER</td>
<td>No</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>Soph</td>
<td>MEMBER</td>
<td>No</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>Junior</td>
<td>SECRETARY</td>
<td>Yes</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>Senior</td>
<td>PRECIDENT</td>
<td>Yes</td>
<td>20</td>
<td>18</td>
</tr>
</tbody>
</table>

### Activity 2

<table>
<thead>
<tr>
<th>Organization / Activity</th>
<th>Activity 2 level</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHESS CLUB</td>
<td>LOCAL</td>
</tr>
</tbody>
</table>

**Description**

*CHESS*

**Participation Details for Activity 2 (Use whole numbers only, no fractions.)**

<table>
<thead>
<tr>
<th>Year</th>
<th>Position(s) Held</th>
<th>Were You Elected?</th>
<th>Hours/week</th>
<th>Weeks/year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh</td>
<td></td>
<td>Not Applicable</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Soph</td>
<td>MEMBER</td>
<td>No</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not Applicable</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not Applicable</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Fact: Roughly 250,000 high-school graduates each year

Guess: 125,000 uses of www.applytexas.org

Guess: 60,000 need to re-order extracurricular activities

Time invested:

\[ 60,000 \times 5 \text{ mins} = 300,000 \text{ mins} \]
Fact: Roughly 250,000 high-school graduates each year
Guess: 125,000 uses of www.applytexas.org
Guess: 60,000 need to re-order extracurricular activities
Time invested:

\[
60,000 \times 5 \text{ mins} = 300,000 \text{ mins} \\
\sim 208 \text{ days}
\]
Back of the envelope estimate

- Fact: Roughly 250,000 high-school graduates each year
- Guess: 125,000 uses of www.applytexas.org
- Guess: 60,000 need to re-order extracurricular activities
- Time invested:

\[
60,000 \times 5 \text{ mins} = 300,000 \text{ mins} \\
\sim 208 \text{ days} \\
> \frac{1}{2} \text{ year}
\]
Fact: Roughly 250,000 high-school graduates each year

Guess: 125,000 uses of www.applytexas.org

Guess: 60,000 need to re-order extracurricular activities

Time invested:

\[ 60,000 \times 5 \text{ mins} = 300,000 \text{ mins} \]
\[ \sim 208 \text{ days} \]
\[ > \frac{1}{2} \text{ year} \]
\[ \sim 2 \text{ developer years} \]
Impact of a nuisance

- ApplyTexas.org is just one little app in one corner of the world, but the same repeats everywhere
  - e-commerce sites
  - travel bookings
  - tax form preparation software
  - “in-house” business applications
  - even high-end desktop applications
Impact of a nuisance

- ApplyTexas.org is just one little app in one corner of the world, but the same repeats everywhere
  - e-commerce sites
  - travel bookings
  - tax form preparation software
  - “in-house” business applications
  - even high-end desktop applications

- A small waste of effort significant when aggregated over a large number of users
Impact of a nuisance

- ApplyTexas.org is just one little app in one corner of the world, but the same repeats everywhere
  - e-commerce sites
  - travel bookings
  - tax form preparation software
  - “in-house” business applications
  - even high-end desktop applications

- A small waste of effort significant when aggregated over a large number of users

- A small waste of effort significant even for one user when repeated in many user interfaces or by repeated use of one
Poor quality of user interfaces contribute to a significant waste of human effort.
Why everything is broken and nobody’s upset

- Users experience low quality in small doses, too small to complain
- An individual user’s reaction to a usability problem
  - grumbling
  - attempt to find a work-around
  - succeed or give up
  - soldier on
- Perceived per user cost of low quality is low
- Per developer cost of eliminating frustration is high(er)
Users experience low quality in small doses, too small to complain

An individual user’s reaction to a usability problem
  - grumbling
  - attempt to find a work-around
  - succeed or give up
  - soldier on

Perceived per user cost of low quality is low

Per developer cost of eliminating frustration is high(er)

This imbalance rewards producing barely passable quality
Users experience low quality in small doses, too small to complain.

An individual user’s reaction to a usability problem:
- grumbling
- attempt to find a work-around
- succeed or give up
- soldier on

Perceived per user cost of low quality is low.

Per developer cost of eliminating frustration is high(er).

This imbalance rewards producing barely passable quality.

Even if this was not the case, programming feature-rich and correct UIs is not easy at all (demo).
Some considerations for the UI programmer:

- Which fields need to be recomputed and to which values after a change
- Should some widgets be disabled or enabled after an interaction
- Indicate that a value is pending if there is a delay
- Keep the UI responsive even though some values are pending
- Keep updates consistent and cancel unnecessary computations in case interactions happen while computation is ongoing
- Invalid inputs should be rejected or indicated somehow
- Helpful error messages should be given to the user, pointing accurately where troublesome values are
- Failed computations by the user interface should be handled, and the reasons communicated through helpful error messages
- Undo/redo
Copy/paste

Reacting to external changes (change of window size, abruptly closing the window)

Support both mouse and keyboard navigation

The UI should support *scripting*
Algorithms for User Interfaces

Ideal

- Developing a high-quality feature-rich GUI is no more expensive than developing a low-quality bare-bones GUI.
Algorithms for User Interfaces

Ideal

- Developing a high-quality feature-rich GUI is no more expensive than developing a low-quality bare-bones GUI.

Approach

- Declarative programming, constraint systems
  - Specify dependencies amongst data in a GUI as a hierarchical multi-way data-flow constraint system
  - A non-incidental real data structure
- GUI behaviors are reusable algorithms over the constraint system data structure
  - updating values, enabling/disabling widgets, scripting, undo/redo, spinners for pending values, responsiveness, pinning values, accurate error messages, ...
Model for UIs: Data with constraints

- Express data and its dependencies as an explicit model
- User change may bring data into an *inconsistent* state
- UI reacts by restoring consistency
Express data and its dependencies as an explicit model
User change may bring data into an *inconsistent* state
UI reacts by restoring consistency
Express data and its dependencies as an explicit model

User change may bring data into an *inconsistent* state

UI reacts by restoring consistency
Express data and its dependencies as an explicit model
User change may bring data into an *inconsistent* state
UI reacts by restoring consistency
Express data and its dependencies as an explicit model
User change may bring data into an *inconsistent* state
UI reacts by restoring consistency
Express data and its dependencies as an explicit model

User change may bring data into an *inconsistent* state

UI reacts by restoring consistency
Express data and its dependencies as an explicit model

User change may bring data into an *inconsistent* state

UI reacts by restoring consistency
Jaakko Järvi
Avoiding Faulty User Interfaces
Feb 11th, 2014 17 / 26
Event handler code for "onChange" event in a view:

1. Update a variable in the constraint system
2. Solve
3. Other views update their values
Model for UIs: Multi-way dataflow constraint system

Event handler code for "onChange" event in a view:

1. Update a variable in the constraint system
2. Solve
3. Other views update their values

Jaakko Järvi
Avoiding Faulty User Interfaces
Feb 11th, 2014
Model for UIs: Multi-way dataflow constraint system

Event handler code for "onChange" event in a view:
1. Update a variable in the constraint system
2. Solve
3. Other views update their values
Event handler code for "onChange" event in a view is trivial:

1. Update a variable in the constraint system
2. Solve
3. Other views update their values
Event handler code for “onChange” event in a view trivial:

1. update a variable in the constraint system
2. solve
3. other views update their values
Incidental Data Structure $\rightarrow$ Explicit Model

![Diagram of data structure relationships]

- Initial Height
- Initial Weight
- Absolute Height
- Absolute Width
- Relative Height
- Relative Width
- Last updated
- Preserve ratio
- Event Handler Constr.
- Event Handler Abs. H.
- Event Handler Abs. W.
- Event Handler Rel. H.
- Event Handler Rel. W.
Incidental Data Structure $\rightarrow$ Explicit Model

![Diagram showing the relationship between Incidental Data Structure and Explicit Model]

- **Incidental Data Structure**
  - Initial Height
  - Initial Weight
  - Absolute Height
  - Absolute Width
  - Relative Height
  - Relative Width
  - Event Handler
  - Event Handler Abs. H.
  - Event Handler Abs. W.
  - Event Handler Rel. H.
  - Event Handler Rel. W.
  - Dialog
  - Last updated
  - Preserve ratio
  - Event Handler Constr.

- **Explicit Model**
  - Initial Height
  - Absolute Height
  - Relative Height (%)
  - Relative Height
  - Absolute Width
  - Relative Width (%)
  - Relative Width
  - Event Handler

Jaakko Järvi
Avoiding Faulty User Interfaces
Feb 11th, 2014
def ChangeCurrentHeightPx(self, event):
    self.LastUpdated = "Height"
    constrained = self.Controls["Constrain"]["GetValue"]
    if constrained: # no matter what the percent & current stay bound together
        # get current height, and compute relative height and place new rel. ht
        height = float(self.Controls["AbsolutePx"]["Height"]["GetValue"])
        pct = height / self.InitialSize["Height"]
        self.Controls["Relative%"]["Height"]["SetValue"](str(pct*100))
        if constrained: # update width & width%
            width = pct * self.InitialSize["Width"]
            self.Controls["AbsolutePx"]["Width"]["SetValue"](str(round(width)))
    width = float(self.Controls["AbsolutePx"]["Width"]["GetValue"])
    self.Controls["Relative%"]["Width"]["SetValue"](str(pct))
    height = width * self.InitialSize["Height"] / 100
    self.Controls["AbsolutePx"]["Height"]["SetValue"](str(round(height)))

def ChangeCurrentWidthPx(self, event):
    self.LastUpdated = "Width"
    constrained = self.Controls["Constrain"]["GetValue"]
    if constrained: # no matter what the percent & current stay bound together
        # get current width, and compute relative width and place new rel. wd
        width = float(self.Controls["AbsolutePx"]["Width"]["GetValue"])
        pct = width / self.InitialSize["Width"]
        self.Controls["Relative%"]["Width"]["SetValue"](str(pct*100))
        if constrained: # update height & height%
            height = pct * self.InitialSize["Height"]
            self.Controls["AbsolutePx"]["Height"]["SetValue"](str(round(height)))
    height = float(self.Controls["AbsolutePx"]["Height"]["GetValue"])
    self.Controls["Relative%"]["Height"]["SetValue"](str(pct))
    width = height * self.InitialSize["Width"] / 100
    self.Controls["AbsolutePx"]["Width"]["SetValue"](str(round(width)))

def ChangeCurrentHeightPct(self, event):
    self.LastUpdated = "Height"
    constrained = self.Controls["Constrain"]["GetValue"]
    if constrained: # no matter what the percent & current stay bound together
        # get current rel. ht, and compute absolute height and place new abs. ht
        height = float(self.Controls["Relative%"]["Height"]["GetValue"])
        cur = height * self.InitialSize["Height"] / 100
        self.Controls["AbsolutePx"]["Height"]["SetValue"](str(round(cur)))
        if constrained: # update width & width%
            width = height * self.InitialSize["Width"] / 100
            self.Controls["AbsolutePx"]["Width"]["SetValue"](str(round(width)))
    width = float(self.Controls["AbsolutePx"]["Width"]["GetValue"])
    self.Controls["Relative%"]["Width"]["SetValue"](str(height))
    height = width * self.InitialSize["Height"] / 100
    self.Controls["AbsolutePx"]["Height"]["SetValue"](str(round(height)))

def ChangeCurrentWidthPct(self, event):
    self.LastUpdated = "Width"
    constrained = self.Controls["Constrain"]["GetValue"]
    if constrained: # no matter what the percent & current stay bound together
        # get current rel. wd, and compute absolute width and place new abs. wd
        width = float(self.Controls["Relative%"]["Width"]["GetValue"])
        cur = width * self.InitialSize["Width"] / 100
        self.Controls["AbsolutePx"]["Width"]["SetValue"](str(round(cur)))
        if constrained: # update height & height%
            height = width * self.InitialSize["Height"] / 100
            self.Controls["AbsolutePx"]["Height"]["SetValue"](str(round(height)))
    height = float(self.Controls["AbsolutePx"]["Height"]["GetValue"])
    self.Controls["Relative%"]["Height"]["SetValue"](str(width))
    width = height * self.InitialSize["Width"] / 100
    self.Controls["AbsolutePx"]["Width"]["SetValue"](str(round(width)))

def ChangeConstrainState(self, event):
    constrained = self.Controls["Constrain"]["GetValue"]
    # If the ratio is constrained, determine which dimension
    # was last updated and update the OTHER dimension.
    # For example: if Height was last updated, use Height as
    # Width's new percent, and update Width's absolute value
    if constrained:
        if self.LastUpdated == "Height": # update width px & %
            pct = float(self.Controls["Relative%"]["Height"]["GetValue"])
            self.Controls["Relative%"]["Width"]["SetValue"](str(pct*100))
            width = pct * self.InitialSize["Width"] / 100
            self.Controls["AbsolutePx"]["Width"]["SetValue"](str(round(width)))
        else: # update width px & %
            pct = float(self.Controls["Relative%"]["Width"]["GetValue"])
            self.Controls["Relative%"]["Height"]["SetValue"](str(pct*100))
            height = pct * self.InitialSize["Height"] / 100
            self.Controls["AbsolutePx"]["Height"]["SetValue"](str(round(height)))

Jaakko Järvi
Avoiding Faulty User Interfaces
Feb 11th, 2014 19 / 26
Code of Incidental Algorithm → Model Declaration

```python
def ChangeCurrentHeightPx(self, event):
    self.LastUpdated = "Height"
    constrained = self.Controls["Constrain"].GetValue()
    if constrained: # update width & widths
        self.Controls["Relative%"["Width"]].SetValue(str(pct*100))
        width = pct * self.InitialSize[self.Width]
        self.Controls["AbsolutePx"["Width"]].SetValue(str(round(width))
    height = float(self.Controls["AbsolutePx"["Height"]].GetValue())
    pct = height / self.InitialSize[self.Height]
    self.Controls["Relative%"["Height"]].SetValue(str(pct*100))
    if constrained: # update width & width%
        self.Controls["Relative%"["Width"]].SetValue(str(pct*100))
        width = pct * self.InitialSize[self.Width]
        self.Controls["AbsolutePx"["Width"]].SetValue(str(round(width))

def ChangeCurrentWidthPx(self, event):
    self.LastUpdated = "Width"
    constrained = self.Controls["Constrain"].GetValue()
    if constrained: # update height & heights
        self.Controls["Relative%"["Height"]].SetValue(str(pct*100))
        height = pct * self.InitialSize[self.Height]
        self.Controls["AbsolutePx"["Height"]].SetValue(str(round(height))
    width = float(self.Controls["AbsolutePx"["Width"]].GetValue())
    pct = height / self.InitialSize[self.Width]
    self.Controls["Relative%"["Width"]].SetValue(str(pct*100))
    if constrained: # update height & height%
        self.Controls["Relative%"["Height"]].SetValue(str(pct*100))
        height = pct * self.InitialSize[self.Height]
        self.Controls["AbsolutePx"["Height"]].SetValue(str(round(height))

def ChangeCurrentHeightPct(self, event):
    self.LastUpdated = "Height"
    constrained = self.Controls["Constrain"].GetValue()
    if constrained: # update width & widths
        self.Controls["Relative%"["Width"]].SetValue(str(height))
        width = height * self.InitialSize[self.Width] / 100
        self.Controls["AbsolutePx"["Width"]].SetValue(str(round(width))
    height = float(self.Controls["AbsolutePx"["Height"]].GetValue())
    cur = height * self.InitialSize[self.Height] / 100
    self.Controls["AbsolutePx"["Height"]].SetValue(str(round(cur))
    if constrained: # update width & width%
        self.Controls["Relative%"["Width"]].SetValue(str(height))
        width = height * self.InitialSize[self.Width] / 100
        self.Controls["AbsolutePx"["Width"]].SetValue(str(round(width))

def ChangeCurrentWidthPct(self, event):
    self.LastUpdated = "Width"
    constrained = self.Controls["Constrain"].GetValue()
    if constrained: # update height & heights
        self.Controls["Relative%"["Height"]].SetValue(str(width))
        height = width * self.InitialSize[self.Height] / 100
        self.Controls["AbsolutePx"["Height"]].SetValue(str(round(height))
    width = float(self.Controls["Relative%"["Width"]].GetValue())
    cur = width * self.InitialSize[self.Width] / 100
    self.Controls["AbsolutePx"["Width"]].SetValue(str(round(cur))
    if constrained: # update height & height%
        self.Controls["Relative%"["Height"]].SetValue(str(width))
        height = width * self.InitialSize[self.Height] / 100
        self.Controls["AbsolutePx"["Height"]].SetValue(str(round(height))
```

sheetimage_resize {
    input:
    initial_width : 5 * 300;
    initial_height : 7 * 300;

    interface:
    preserve_ratio : true;
    absolute_width : initial_width;
    absolute_height : initial_height;
    relative_width; relative_height;

    logic:
    relate {
        relative_height <== absolute_height * 100 / initial_height;
        absolute_height <== relative_height * initial_height / 100;
    }
    relate {
        relative_width <== absolute_width * 100 / initial_width;
        absolute_width <== relative_width * initial_width / 100;
    }
    when (preserve_ratio) relate {
        relative_width <== relative_height;
        relative_height <== relative_width;
    }

    }
sheet image_resize {
    input:
        initial_width : 5 * 300;
        initial_height : 7 * 300;
    interface:
        preserve_ratio : true;
        absolute_width : initial_width;
        absolute_height : initial_height;
        relative_width; relative_height;
    logic:
        relate {
            absolute_height <= relative_height * initial_height / 100;
            relative_height <= absolute_height * 100 / initial_height;
        }
        relate {
            absolute_width <= relative_width * initial_width / 100;
            relative_width <= absolute_width * 100 / initial_width;
        }
        when (preserve_ratio) relate {
            relative_width <= relative_height;
            relative_height <= relative_width;
        }
}
Key observation

Reifying the dependencies enables reusable GUI algorithms.
Key observation

Reifying the dependencies enables reusable GUI algorithms.

- Examples:
  - can a variable impact an output?
  - is a variable pending?
Key observation

Reifying the dependencies enables reusable GUI algorithms.

- Examples:
  - can a variable impact an output?
  - is a variable pending?
Experiences adopting property models

- UI behaviors included
  - Maintaining consistency (updating widget values)
  - Widget enablement/disablement
  - Command activation/deactivation
  - Scripting
- Code reduction of 8—10 to one in statement counts
- Improved quality
  - Fewer defects
  - Consistency among different user interface
  - More features
- Anecdote: impact on a single dialog’s event handling and scripting code
  - Before: 781 statements, 5 known logic defects
  - After: 46 statements, no known defects
Experiment

- Rewriting user interface code for a major desktop application
- Four teams of roughly three engineers each
  - Three teams (AE1–AE3) used the declarative approach
  - Fourth team (TF) a modern vendor-supplied object-oriented UI framework
- Each tasked with rewriting a large number of dialogs and palettes
Results: Productivity

- AE1–AE3 teams
  - completed roughly 75 dialogs and palettes
  - 50 more under way

- TF team
  - completed fewer than 10 altogether
Conclusion

- Programming event-handlers manually is very difficult
- Unrealistic to hope for correct, responsive, feature-rich user interfaces
Programming event-handlers manually is very difficult

Unrealistic to hope for correct, responsive, feature-rich user interfaces

Through careful study of commonalities in UI behavior, it is possible to capture user interface behavior as reusable algorithms

Quality and features can be free