Algorithms for User Interfaces

Jaakko Järvi

Texas A&M University Computer Science and Engineering Parasol Lab

April 3, 2012

Outline

Story of why algorithms matter in programming

or a promise of never having to write a GUI event handler again

Outline

- Story of why algorithms matter in programming
- > or a promise of never having to write a GUI event handler again





Why is software like this?

000	Resiz	e Image		
Initial Height	1500	Initial Width	2100	
Absolute Height	1500.0	Absolute Width	2100	
Relative Height (%)	100.0	Relative Width (%)	100.0	1
Preserve ratio		Cancel	ОК	



del ChangeCurrentHeightYx[self, event]: self_Lastic/dated - "Height" constrained - self Controls[Constrain"],Get(Value) # on onatter what the percent & current stay bound together # get current height, and compute relative height and place new rel. ht height - flagt(self-Controls]?AbsouldPer]THeight],GetMaue()) pct - height / self_Lonstois]?AbsouldPer]THeight],GetMaue()) pct - height / self_Lonstois]?AbsouldPer]THeight],GetMaue())

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)))

del ChangeCurrentWidthPkjeelt, event): self_astLoidad= - Width' constrained - self.Controls[Constrain],GetValue() # montater with the percent & current stay bound together # get current width, and compute relative width and place new rel. wid height = floatistic Controls[AbsoulterP_]"Width[PetValue()) pcl = height / self.InitiaSizejself.Width] self.Controls[Petvalue()],SetValue(str(pcl-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() # 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]"relatives"[]"Height[",GetValue()] cur = height = self.initialSize[self.Height] (100 self.Controls]" AbsolutePs:("II"Height]"SelfValue(strfround(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() # on matter what the percent & current stay bound together # get current reti. wd. and compute absolute width and place new abs. wd width - float get Controls/Featiwers/TWddth/SelfValue()) cur - width - self.initialSizejeet/Wddth/ 100 self.Controls/Featiwers/TSelfValue(strycond(curr))) if constrained: # update height & height% self.Controls["Relative%:"[]"Height].SetValue(str(width)) height - width - self.InitialSize[self.Height] / 100 self.Controls["Absolute?x:[]"Height].SetValue(str(round(height)))

def ChangeConstrainState(self, event):

constrained = self.Controls["Constrain"].GetValue() # If the ratio is constrained, determine which dimension

If the ratio is constrained, determine which dimension # was last updated and update the OTHER dimension.

was last updated and update the OTHEH dimension. # For example: if Height was last updated, use Height as

Width's new percent, and update Width's absolute value

if constrain

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["Relative%"]"Width",SetValue(str(round(width)))

else: # update width px & % pct = float(self.Controls!"Relative%")["Width"].GetValue())

pct = float(sell.Controls["Relative%"]["Width"].GetValue()) sell.Controls["Relative%"]["Height"].SetValue(str(pct)) height = pct - sell.InitialSize[sell.Height] / 100 sell.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

000	Resize	e Image		
Initial Height	1500	Initial Width	2100	
Absolute Height	1500.0	Absolute Width	2100	
Relative Height (%)	100.0	Relative Width (%)	100.0	
Preserve ratio		Cancel	ОК	



- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vasts amounts of well tested and proven code routinely reused
 - GUI components, delivering events, rendering, capturing interaction
 - Example: a typical TextBox widget: 100 methods, recognizes
 > 200 events
- Compositions are not reusable
 - \Rightarrow ad-hoc code, defects, inconsistent behavior, costly development
- Incidental data structures arise from a network of objects
- Incidental algorithms arise from the concert of localized actions
- Minimal requirement for reuse: understandable model
 - Not satisfied by incidental data structures and algorithms

Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0
Preserve ratio		Cancel	ОК

Resize Image



- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vasts amounts of well tested and proven code routinely reused
 - GUI components, delivering events, rendering, capturing interaction
 - Example: a typical TextBox widget: 100 methods, recognizes
 > 200 events
- Compositions are not reusable
 - \Rightarrow ad-hoc code, defects, inconsistent behavior, costly development
- Incidental data structures arise from a network of objects
- Incidental algorithms arise from the concert of localized actions
- Minimal requirement for reuse: understandable model
 - Not satisfied by incidental data structures and algorithms

Motivation	
000	Resize Image

Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0
Preserve ratio		Cancel	ОК



- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vasts amounts of well tested and proven code routinely reused
 - GUI components, delivering events, rendering, capturing interaction
 - Example: a typical TextBox widget: 100 methods, recognizes
 > 200 events
- Compositions are not reusable
 - \Rightarrow ad-hoc code, defects, inconsistent behavior, costly development
- Incidental data structures arise from a network of objects
- Incidental algorithms arise from the concert of localized actions
- Minimal requirement for reuse: understandable model
 - Not satisfied by incidental data structures and algorithms

Motivati	on	
000		Resize Image
Initial Height	1500	Initial

Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0
Preserve ratio		Cancel	ОК



- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vasts amounts of well tested and proven code routinely reused
 - GUI components, delivering events, rendering, capturing interaction
 - Example: a typical TextBox widget: 100 methods, recognizes
 > 200 events
- Compositions are not reusable
 - \Rightarrow ad-hoc code, defects, inconsistent behavior, costly development
- Incidental data structures arise from a network of objects
- Incidental algorithms arise from the concert of localized actions
- Minimal requirement for reuse: understandable model
 - Not satisfied by incidental data structures and algorithms

Μ	lotivation	
	000	

Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0
Preserve ratio		Cancel	ОК

Resize Image



- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vasts amounts of well tested and proven code routinely reused
 - GUI components, delivering events, rendering, capturing interaction
 - Example: a typical TextBox widget: 100 methods, recognizes
 > 200 events
- Compositions are not reusable
 - \Rightarrow ad-hoc code, defects, inconsistent behavior, costly development
- Incidental data structures arise from a network of objects
- Incidental algorithms arise from the concert of localized actions
- Minimal requirement for reuse: understandable model
 - Not satisfied by incidental data structures and algorithms

Software is forever doomed!

Given a sorted array $A[0] \le A[1] \le A[n-1]$, we want to determine if a given element T is in the array. Binary search solves the problem by keeping track of a range within the array in which T must be if it is anywhere in the array. Initially the range is the entire array. The range is shrunk by comparing its middle element to T, and then discarding half the range. The process continues until T is found, or until the range in which it must lie is known to be empty. In an *n*-element table, the search uses roughly $\log_2(n)$ comparisons.

Software is forever doomed!

```
int* binary_search(int* first, int* last, int x) {
  while (first != last) {
    int* middle = first + (last - first) / 2;
    if (*middle < x) first = middle + 1;
    else last = middle;
  }
  return first;
}</pre>
```

Cancel that, programming is not forever doomed after all

- The problem: UI related code is
 - bloated and buggy
 - for example, Adobe's desktop applications, event handling is estimated to account for a third of the code and over half of the defects
 - full of incidental data structures and algorithms
- An approach for improving the status quo
 - ▶ To understand the commonalities that exist in event-handling code
 - To define a model that captures these commonalities
 - ► To apply
 - replace incidental data structures with explicit data structures
 - replace incidental algorithms with explicit reusable algorithm
- Result: substantial increase in reuse, programming productivity, software correctness and quality

Cancel that, programming is not forever doomed after all

- The problem: UI related code is
 - bloated and buggy
 - for example, Adobe's desktop applications, event handling is estimated to account for a third of the code and over half of the defects
 - full of incidental data structures and algorithms
- An approach for improving the status quo
 - > To understand the commonalities that exist in event-handling code
 - To define a model that captures these commonalities
 - To apply
 - replace incidental data structures with explicit data structures
 - replace incidental algorithms with explicit reusable algorithm
- Result: substantial increase in reuse, programming productivity, software correctness and quality

Outline

Motivation

Command Parameter Synthesis

Property Models as Multi-way Dataflow Constraint Systems

What was achieved

Experience and Conclusions

Dialogs serve to

000	Resize	Image	
Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0
Preserve ratio		Cancel	ОК

Dialogs serve to

assist the user in selecting values for parameters to some command

000	Resize	Image	
Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0
Preserve ratio		Cancel	ОК

Command interested in only a few values

Dialogs serve to

assist the user in selecting values for parameters to some command



Command interested in only a few values

Dialog may provide more values than necessary for assistance

Dialogs serve to

000	Resize	Image	
Initial Height	1500	Initial Width	2100
Absolute Height		Absolute Width	
Relative Height (%)	200.0	Relative Width (%)	
Preserve ratio		Cancel	ОК

- Command interested in only a few values
 - Dialog may provide more values than necessary for assistance
- After the user edits a value,
 - The dialog is inconsistent

Dialogs serve to

000	Resize	Image	
Initial Height	1500	Initial Width	2100
Absolute Height		Absolute Width	
Relative Height (%)	200.0	Relative Width (%)	
Preserve ratio		Cancel	ОК

- Command interested in only a few values
 - Dialog may provide more values than necessary for assistance
- After the user edits a value,
 - The dialog is inconsistent
- Then it tries to restore consistency

Dialogs serve to



- Command interested in only a few values
 - Dialog may provide more values than necessary for assistance
- After the user edits a value,
 - The dialog is inconsistent
- Then it tries to restore consistency

Dialogs serve to



- Command interested in only a few values
 - Dialog may provide more values than necessary for assistance
- After the user edits a value,
 - The dialog is inconsistent
- Then it tries to restore consistency

Dialogs serve to



- Command interested in only a few values
 - Dialog may provide more values than necessary for assistance
- After the user edits a value,
 - The dialog is inconsistent
- Then it tries to restore consistency

Outline

Motivation

Command Parameter Synthesis

Property Models as Multi-way Dataflow Constraint Systems

What was achieved

Experience and Conclusions







- Variables ...
- tied together by constraints ...



- Variables ...
- tied together by constraints ...
 - $\text{Height}_{\text{Absolute}} = \text{Height}_{\text{Initial}} \cdot \left(\frac{\text{Height}_{\text{Relative}}}{100}\right)$



- Variables ...
- tied together by constraints ...
 - $\text{Height}_{\text{Absolute}} = \text{Height}_{\text{Initial}} \cdot (\frac{\text{Height}_{\text{Relative}}}{100})$
- each of which can be satisfied by any of a number of methods



- Variables ...
- tied together by constraints ...
 - $\text{Height}_{\text{Absolute}} = \text{Height}_{\text{Initial}} \cdot \left(\frac{\text{Height}_{\text{Relative}}}{100}\right)$
- each of which can be satisfied by any of a number of methods
 - a: absolute_height = initial_height * relative_height / 100;
 - b: relative_height = (absolute_height / initial_height) * 100;



Restoring consistency is now just solving the system



- Restoring consistency is now just solving the system
- Solution defines a dataflow



- Restoring consistency is now just solving the system
- Solution defines a dataflow
 - Selection of methods (in order) such that
 - all constraints enforced
 - no two methods output to same variable



- Restoring consistency is now just solving the system
- Solution defines a dataflow
 - Selection of methods (in order) such that
 - all constraints enforced
 - no two methods output to same variable



- Restoring consistency is now just solving the system
- Solution defines a dataflow
 - Selection of methods (in order) such that
 - all constraints enforced
 - no two methods output to same variable



- Restoring consistency is now just solving the system
- Solution defines a dataflow
 - Selection of methods (in order) such that
 - all constraints enforced
 - no two methods output to same variable



- Restoring consistency is now just solving the system
- Solution defines a dataflow
 - Selection of methods (in order) such that
 - all constraints enforced
 - no two methods output to same variable
 - ▶ e.g. a, $e \rightarrow c$



► Programmer only defines relations and their methods, not which method to execute and when ⇒ often multiple solutions



► Programmer only defines relations and their methods, not which method to execute and when ⇒ often multiple solutions



- ► Programmer only defines relations and their methods, not which method to execute and when ⇒ often multiple solutions
 - Need a way to order them



- ► Programmer only defines relations and their methods, not which method to execute and when ⇒ often multiple solutions
 - Need a way to order them
- In general, want to prefer methods that change older values



Initial Height	1
Initial Width	2
Relative Height	3
Absolute Height	4
Relative Width	5
Absolute Width	6

- ► Programmer only defines relations and their methods, not which method to execute and when ⇒ often multiple solutions
 - Need a way to order them
- In general, want to prefer methods that change older values
- Priorities



Initial Height	1
Initial Width	2
Relative Height	3
Absolute Height	4
Relative Width	5
Absolute Width	6

- ► Programmer only defines relations and their methods, not which method to execute and when ⇒ often multiple solutions
 - Need a way to order them
- In general, want to prefer methods that change older values
- Priorities = Hierarchical Stay Constraints



Initial Height	1
Initial Width	2
Relative Height	3
Absolute Height	4
Relative Width	5
Absolute Width	6

- ► Programmer only defines relations and their methods, not which method to execute and when ⇒ often multiple solutions
 - Need a way to order them
- In general, want to prefer methods that change older values
- Priorities = Hierarchical Stay Constraints
 - Stay constraint = does nothing, so its variable stays the same



Initial Height	1
Initial Width	2
Relative Height	3
Absolute Height	4
Relative Width	5
Absolute Width	6

- ► Programmer only defines relations and their methods, not which method to execute and when ⇒ often multiple solutions
 - Need a way to order them
- In general, want to prefer methods that change older values
- Priorities = Hierarchical Stay Constraints
 - Stay constraint = does nothing, so its variable stays the same
 - Hierarchy = groups of constraints with certain strength

Explicit Algorithm for Command Parameter Synthesis

- Each UI element has a variable in a constraint system
- Event handling code becomes auto-generated boilerplate
 - Value modification generates a request to the constraint system to modify one variable and its priority, and solve
 - At all times, the UI element shows the value of the variable in the constraint system

Outline

Motivation

Command Parameter Synthesis

Property Models as Multi-way Dataflow Constraint Systems

What was achieved

Experience and Conclusions

Incidental Data Structure \rightarrow Explicit Model

Incidental Data Structure \rightarrow Explicit Model



Incidental Data Structure \rightarrow Explicit Model





Code of Incidental Algorithm \rightarrow Model Declaration

del ChangeCurrentHeightPix(self, event): self_astic_bated = "Height" constrained = self.Control(PConstrain]", GetValue() # o matter what the percent" acurent stay bound together # got current height, and compute relative height and place new rel. ht height = float(actic_control(PAcostuber)"]"HeightPix(actube()) pd = height / self.thisSizcigatHeight] self.control(PAcostuber)"["HeightPix(actube())] pd = height / self.thisSizcigatHeight] self.control(PAcostube()); Selvalue()(pt(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)))

del ChangeCurrontWichP(soft, event): sell Lastitylateid = "Wich" constrained = self.Controls[Constrain],GetValue() # no matter watte persont & current stay bound together # get current wicht, and compute relative wicht and piace new rel. with height = Medical:Controls[#NoculterP]"WichT],GetValue()) pct = height / self.LinklaStogetLWidh] self.Controls[#NoculterP]"WichT],SetValue(Stript-100))

If constrained: # update height & height% self.Controls["Relative%"]["Height"].SetValue(str(pct+100)) height = pct - self.InitilSize[self.Height] self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

del ChangeCurrentHeightFictat, eventi: sell_astibuted - Height constrained - sell Control(Constain) (GetVabae() F no matter what the percent & current stay bound together # get current rei, ht, and compute absolute height and piace new sts. ht height = Mostic Control(Fabatue()'''_Height() (GetVabae()) cur = height - sell theight() folo sell.Control(Fabatue)'''_Fabe(ht)(-Selvabae()'''_Fabe(ht)(sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabatue)''_Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabatue)'''_Fabatue)''_Fabatue)'''_Fabatu sell.Controls["Relative%"]["Width"].SetValue(str(height)) width = height = setLihitalSzejcell Width] / 100 sell.Controls["Absolute*","[[Width"].SetValue(str(round(width))]) det ChanosCurrentWidthPctfsell.event1:

te oriengeourient real experimentation of the second of th

If constrained: # update height & height% sell Controls["Relative%"]["Height"]. SetValue(str(width)) height = width - setLinitalSize[setLHeight] / 100 sell Controls["AbsolutePx"]["Height"]. SetValue(str(round(height)))

del ChangeConstrainStatis(e), event): constraind = a ell'Controis(Constrain) GetValue) # If the ratio is constrained, determine which dimension # vans last updated and update the OTHER dimension. # For example: If Height was last updated, use Height as # Width's new percent, and updated Width's absolute value If self.LastUpdated == "Height': # updated width va & % pc = foreigel.Controls("Federative")("Height', GetValue())

pct = floatiset Lonnoid = Palative's ["Height].setValue()) self.Control(#Ralaive's ["Work] SetValue(strippt)) widh = pct = self.InitialSteejeelf Width) / 100 self.Control(#Rasilve's ["Width"].SetValue(strippt)) eter = floatiset(Control(#Ralaive's ["Width"].GetValue(strippt)) self.Control(#Relaive's ["Height"].SetValue(strippt)) self.Control(#Relaive's ["Height"].SetValue(strippt)) self.Control(#Relaive's ["Height"].SetValue(strippt))

self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

If constrained: # update width & width%

Code of Incidental Algorithm \rightarrow Model Declaration

del ChangeCurrentHeightP(sell, event): sell_astit/patiend = 'Height' constrained = sell.Controls[Constain1,GetValue() # omatter what the percent & current stay bound together # get current height, and compute relative height and place new rel. ht height = floatistic.Controls[7:Kostuber]T[=Height].GetValue()) pct = height / sell.InitialSice[sell.Height] sell.Controls[7:Kostuber]T[=Height].GetValue())

If constrained: # update width & width% self.Controls["Relative%", "["Width"].SetValue(str(pct+100)) width = pct + self.InitialSize[self.Width] self.Controls["AbsoluteFx"]["Width"].SetValue(str(round(width)))

del ChangeCurrontWichP(soft, event): sell Lastitylateid = "Wich" constrained = self.Controls[Constrain],GetValue() # no matter watte persont & current stay bound together # get current wicht, and compute relative wicht and piace new rel. with height = Medical:Controls[#NoculterP]"WichT],GetValue()) pct = height / self.LinklaStogetLWidh] self.Controls[#NoculterP]"WichT],SetValue(Stript-100))

If constrained: # update height & height% self.Controls["Relative%"]["Height"].SetValue(str(pct+100)) height = pct - self.InitilSize[self.Height] self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

del ChangeCurrentHeightFictat, eventi: sell_astibuted - Height constrained - sell Control(Constain) (GetVabae() F no matter what the percent & current stay bound together # get current rei, ht, and compute absolute height and piace new sts. ht height = Mostic Control(Fabatue()'''_Height() (GetVabae()) cur = height - sell theight() folo sell.Control(Fabatue)'''_Fabe(ht)(-Selvabae()'''_Fabe(ht)(sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)'''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabatue)''_Fabatue)''_Fabe(ht)(-Sell.Control(Fabatue)''_Fabatue)'''_Fabatue)''_Fabatue)'''_Fabatu self.Controls["Relative%"]["Width"].SetValue(str(height)) width = height + self.InitialSize[self.Width] / 100 self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

- del ChangeGurrenWethPelgisalt, event): enll Lastit, botten = Weth? constained - self. Controllogi/Constrain"], GetValual) # on matter with the procent & current stay bound together # get current rel. wd. and compute absolute with and place new abs. wd with = floatgate: Controligi Relativity" [Weth?] (SatValua(i)) cur = widh - self.initialSisget/Weth] / 100 self. Controligi Relativity [SatValua(stricround(curr))))
- If constrained: # update height & height% sell Controls["Relative%"]["Height"]. SetValue(str(width)) height = width - setLinitalSize[setLHeight] / 100 sell 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". # update width px & % pct = float(self.Controls["Relative%"]"Height",GelValue()) self.Controls["Relative%"]"Width",SelValue(stripcti)) width = pct - self.InfallsZelgelt.Width",100 self.Controls["AbsolutePs"]"Width",SelValue(stripcund(width))) else.# update width px & % pct = float(self.Controls["Relative%"]"Width",GelValue())
 - pct = most[setr.Controls["Heatwess"]["Weight"].SetValue(str(pct)) height = pct * setf.initialSize(setF.Height] / 100 setf.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

sheet image_resize { input:

initial_width : 5 • 300; initial_height : 7 • 300;

preserve ratio : true:

preserve_rabo : 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;

Declarative Specification of Command Parameter Synthesis

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;</pre>
```



Declarative Specification of Command Parameter Synthesis

```
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;
```



Declarative Specification of Command Parameter Synthesis

```
sheet image resize {
 input:
    initial width : 5 * 300:
   initial height : 7 * 300;
 interface:
   preserve ratio : true:
                                                                                       Absolute
                                                                                                               Absolute
   absolute width : initial width;
                                                                               3
                                                                                                                          <del>~</del>6)
                                                                                        Height
                                                                                                                Width
   absolute height : initial height;
   relative width: relative height:
 loaic:
   relate {
                                                                          Initia
                                                                                                                              Initial
                                                                          Height
                                                                                                                              Width
     absolute height <== relative height * initial height / 100: // a
     relative height <== absolute height * 100 / initial height: // b
   relate {
                                                                                       Relative
                                                                                                               Relative
     absolute width <== relative width * initial width / 100: // c
                                                                                                                           ~(5)
                                                                                      Height (%)
                                                                                                              Width (%)
     relative width <== absolute width * 100 / initial width; // d
   when (preserve ratio) relate {
     relative width <== relative height: // e
     relative height <== relative width; // f
```

Algorithms for User Interfaces

- Before, every new feature required more spaghetti (incidental) code, specific to each dialog
- Now, each new feature can be defined as a reusable algorithm in a library



Scripting

A script is a recorded sequence of commands

- e.g. remove red-eye, skin blemishes, extra weight
- What do we record from our model as part of the script?
- Remember that probably not every value is useful
 - Some are provided by the document
 - Some are provided by the user
- Only want to capture what the user intended

Scripting

- A script is a recorded sequence of commands
 - e.g. remove red-eye, skin blemishes, extra weight
- What do we record from our model as part of the script?
- Remember that probably not every value is useful
 - Some are provided by the document
 - Some are provided by the user
- Only want to capture what the user intended

Scripting

- A script is a recorded sequence of commands
 - e.g. remove red-eye, skin blemishes, extra weight
- What do we record from our model as part of the script?
- Remember that probably not every value is useful
 - Some are provided by the document
 - Some are provided by the user
- Only want to capture what the user intended

Capturing the User's Intent



Command looks at Absolute Height, Absolute Width,

but what we wanted to change is Relative Height

Capturing the User's Intent



Command looks at Absolute Height, Absolute Width,

but what we wanted to change is Relative Height

Capturing the User's Intent



- Command looks at Absolute Height, Absolute Width,
- but what we wanted to change is Relative Height

Outline

Motivation

Command Parameter Synthesis

Property Models as Multi-way Dataflow Constraint Systems

What was achieved

Experience and Conclusions

Experiences

- Early experience deploying our approach for command parameter synthesis at Adobe
 - Code reductions of a factor of 8 to 10
 - Fewer defects
 - Consistency among user interfaces

Experiment

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

Results: Productivity

- AE1–AE3 teams combined completed roughly 75 dialogs and palettes, with 50 more underway
- TF team completed fewer than 10 altogether

Results: Defect Count



Future Directions

- Opportunities for user interfaces using property models
 - Recently worked on algorithms for enabling/disabling
 - Presets and defaults will follow
 - Perfecting the model for command parameter synthesis
- Incidental structures present in many areas of software
 - Want to know how the approach generalizes
 - Currently developing ideas about applying the declarative approach/constraint systems to other kinds of document modeling