

# Declarative User Interfaces with Property Models

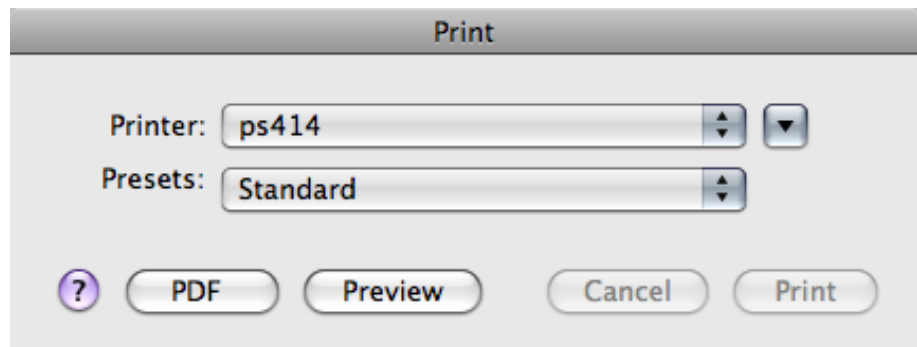
Jaakko Järvi   Mat Marcus   Sean Parent

John Freeman   Jacob N. Smith

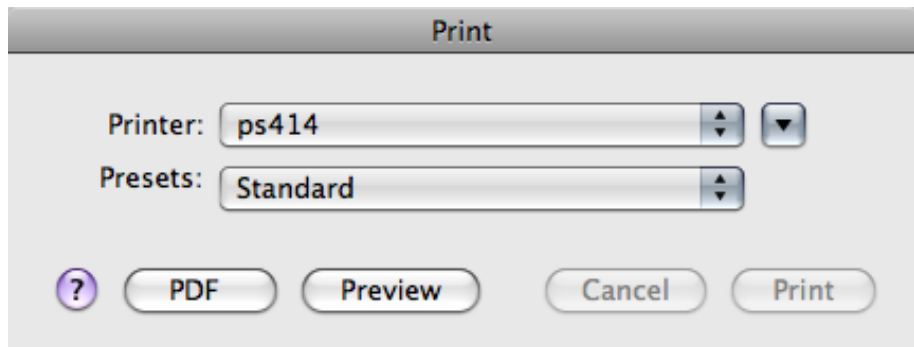
Texas A&M University   Adobe Systems, Inc.

April 13, 2009

## Motivation

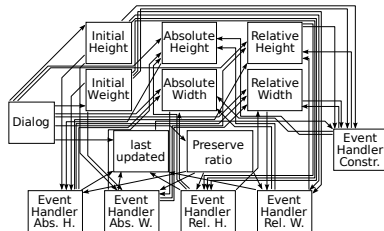
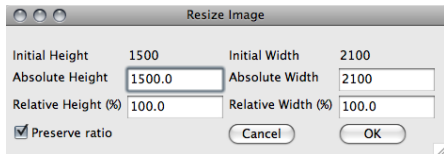


# Motivation



Why is software like this?

# Motivation



```
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%
    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()
    # no matter what the percent & current stay bound together
    # get current width, and compute relative width and place new rel. wd
    height = 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()
    # 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%
    self.Controls["Relative%"]["Width"].SetValue(str(height))
    width = height * self.InitialSize[self.Width] / 100
    self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
```

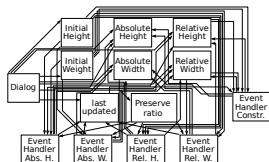
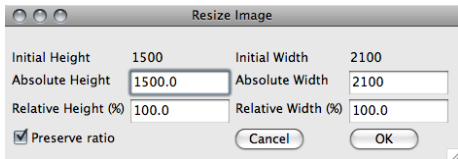
```
def ChangeCurrentWidthPct(self, event):
    self.LastUpdated = "Width"
    constrained = self.Controls["Constrain"].GetValue()
    # no matter what the percent & current stay bound together
    # get current rel. wd, and compute absolute width and place new abs. wd
    width = float(self.Controls["Relative%"]["Width"].GetValue())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

```
if constrained: # update height & height%
    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:
```

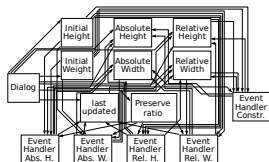
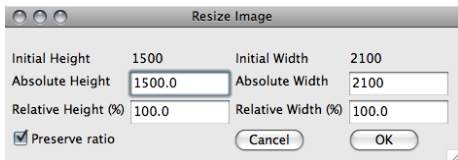
```
if self.LastUpdated == "Height": # update width px & %
    pct = float(self.Controls["Relative%"]["Height"].GetValue())
    self.Controls["Relative%"]["Width"].SetValue(str(pct))
    width = pct * self.InitialSize[self.Width] / 100
    self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
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)))
```

# Motivation



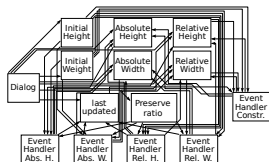
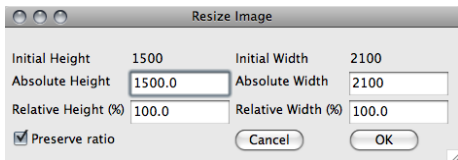
- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vast amounts of well tested and proven code routinely reused
  - GUI components, delivering events, rendering, capturing user's actions
- Compositions are not reusable
  - ⇒ ad-hoc solutions, defects, inconsistent behavior, costly development
- Incidental data structures that arise from a network of objects
- Incidental algorithms that arise from the concert of localized actions
- Minimal requirement for reuse: understandable model
  - Not satisfied by incidental data structures and algorithms

# Motivation



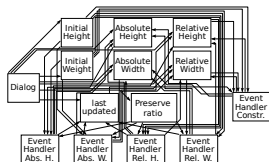
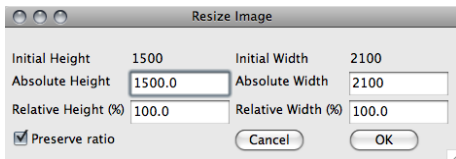
- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vast amounts of well tested and proven code routinely reused
  - GUI components, delivering events, rendering, capturing user's actions
- Compositions are not reusable
  - ⇒ ad-hoc solutions, defects, inconsistent behavior, costly development
- Incidental data structures that arise from a network of objects
- Incidental algorithms that arise from the concert of localized actions
- Minimal requirement for reuse: understandable model
  - Not satisfied by incidental data structures and algorithms

# Motivation



- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vast amounts of well tested and proven code routinely reused
  - GUI components, delivering events, rendering, capturing user's actions
- Compositions are not reusable
  - ⇒ ad-hoc solutions, defects, inconsistent behavior, costly development
- Incidental data structures that arise from a network of objects
- Incidental algorithms that arise from the concert of localized actions
- Minimal requirement for reuse: understandable model
  - Not satisfied by incidental data structures and algorithms

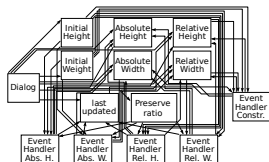
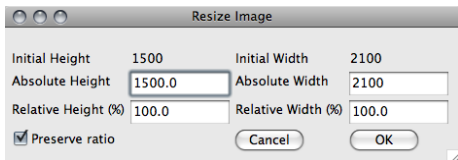
# Motivation



- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vast amounts of well tested and proven code routinely reused
  - GUI components, delivering events, rendering, capturing user's actions
- Compositions are not reusable
  - ⇒ ad-hoc solutions, defects, inconsistent behavior, costly development
- **Incidental data structures** that arise from a network of objects
- **Incidental algorithms** that arise from the concert of localized actions
- Minimal requirement for reuse: **understandable model**
  - Not satisfied by incidental data structures and algorithms



# Motivation



- Reuse is a proven and successful route to improve quality of software, and increase programmer productivity
- Vast amounts of well tested and proven code routinely reused
  - GUI components, delivering events, rendering, capturing user's actions
- Compositions are not reusable
  - ⇒ ad-hoc solutions, defects, inconsistent behavior, costly development
- **Incidental data structures** that arise from a network of objects
- **Incidental algorithms** that arise from the concert of localized actions
- Minimal requirement for reuse: **understandable model**
  - Not satisfied by incidental data structures and algorithms

# Approach

- The Problem

- Complex,
  - In Adobe's desktop applications, event handling is estimated to account for a third of the code...
- Buggy,
  - ...and over half of the defects
- Incidental data structures and algorithms

- Our Approach

- To understand
  - the role of a user interface
  - the commonalities that exist in event-handling code
- To define
  - a model that captures these commonalities
- To apply
  - and gain substantial increase in reuse

# Approach

- The Problem

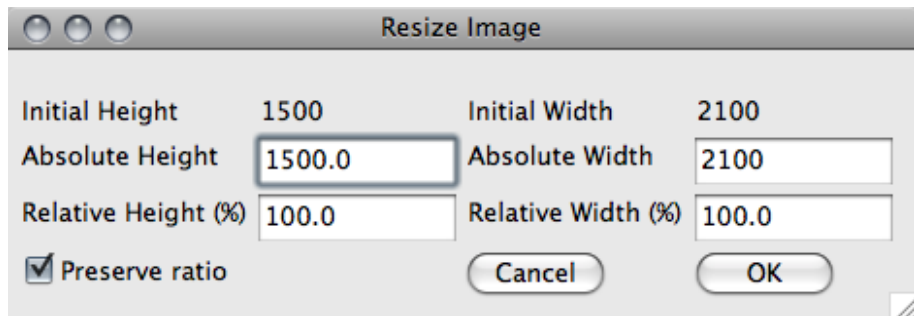
- Complex,
  - In Adobe's desktop applications, event handling is estimated to account for a third of the code...
- Buggy,
  - ...and over half of the defects
- Incidental data structures and algorithms

- Our Approach

- To understand
  - the role of a user interface
  - the commonalities that exist in event-handling code
- To define
  - a model that captures these commonalities
- To apply
  - and gain substantial increase in reuse

## Understanding UIs: *Command Parameter Synthesis*

- Dialogs serve to assist the user in selecting values for parameters to some command



A screenshot of a 'Resize Image' dialog box. The dialog has a title bar with three window control buttons (red, yellow, green) on the left and the title 'Resize Image' in the center. The main area contains four input fields arranged in a 2x2 grid. The first row shows 'Initial Height' with the value '1500' and 'Initial Width' with the value '2100'. The second row shows 'Absolute Height' with a text box containing '1500.0' and 'Absolute Width' with a text box containing '2100'. Below these, the third row shows 'Relative Height (%)' with a text box containing '100.0' and 'Relative Width (%)' with a text box containing '100.0'. At the bottom left, there is a checked checkbox labeled 'Preserve ratio'. At the bottom right, there are two buttons: 'Cancel' and 'OK'. The dialog box has a standard Mac OS X style with a light gray background and rounded corners.

Parameter	Value
Initial Height	1500
Initial Width	2100
Absolute Height	1500.0
Absolute Width	2100
Relative Height (%)	100.0
Relative Width (%)	100.0

☒ Preserve ratio

Cancel OK

## Understanding UIs: *Command Parameter Synthesis*

- Dialogs serve to assist the user in selecting values for parameters to some command

Initial Height 1500 Initial Width 2100

Absolute Height 1500.0 Absolute Width 2100

Relative Height (%) 100.0 Relative Width (%) 100.0

☒ Preserve ratio

Cancel OK

- Command interested in only a few values

## Understanding UIs: *Command Parameter Synthesis*

- Dialogs serve to assist the user in selecting values for parameters to some command

Initial Height 1500 Initial Width 2100

Absolute Height 1500.0 Absolute Width 2100

Relative Height (%) 100.0 Relative Width (%) 100.0

☒ Preserve ratio

Cancel OK

- Command interested in only a few values
  - Dialog may provide more values than necessary for assistance

## Understanding UIs: *Command Parameter Synthesis*

- Dialogs serve to assist the user in selecting values for parameters to some command

Initial Height: 1500 Initial Width: 2100

Absolute Height: Absolute Width:

Relative Height (%): 200.0 Relative Width (%):

☒ Preserve ratio

Cancel OK

- 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

## Understanding UIs: *Command Parameter Synthesis*

- Dialogs serve to assist the user in selecting values for parameters to some command

Initial Height: 1500 Initial Width: 2100

Absolute Height: Absolute Width:

Relative Height (%): 200.0 Relative Width (%):

☒ Preserve ratio

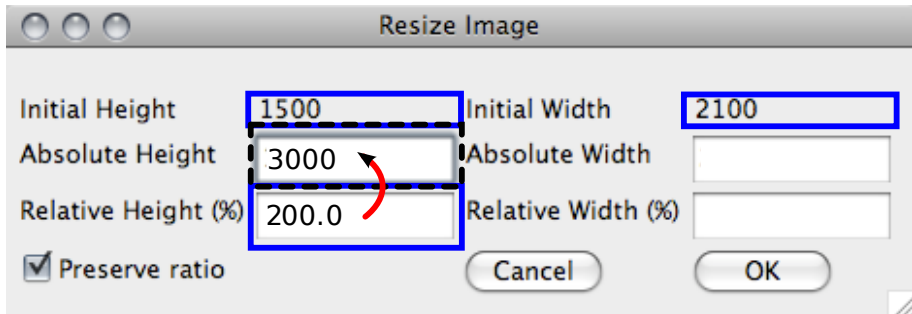
Cancel OK

- 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



## Understanding UIs: *Command Parameter Synthesis*

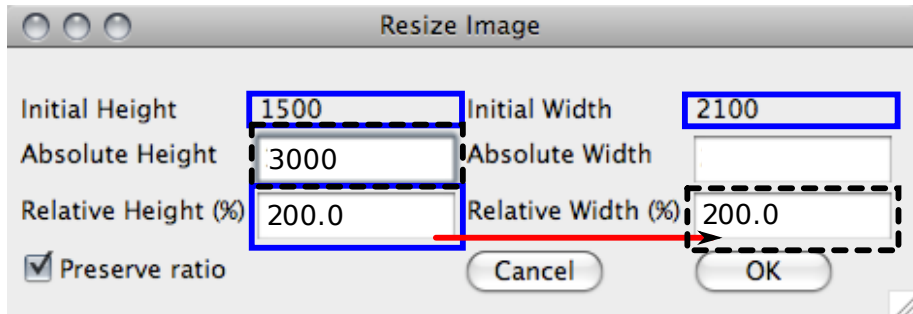
- Dialogs serve to assist the user in selecting values for parameters to some command



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

## Understanding UIs: *Command Parameter Synthesis*

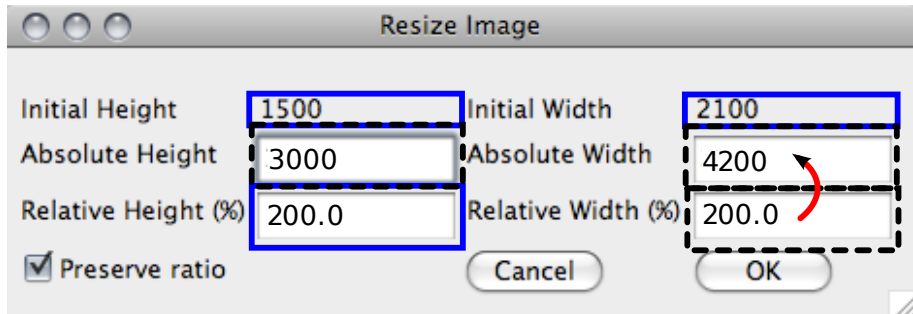
- Dialogs serve to assist the user in selecting values for parameters to some command



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

## Understanding UIs: *Command Parameter Synthesis*

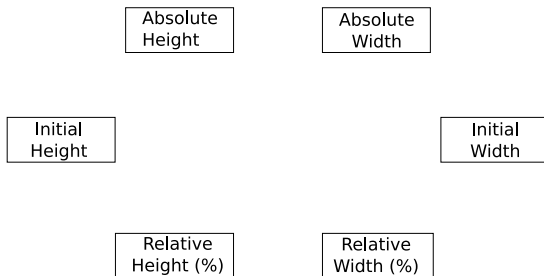
- Dialogs serve to assist the user in selecting values for parameters to some command



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

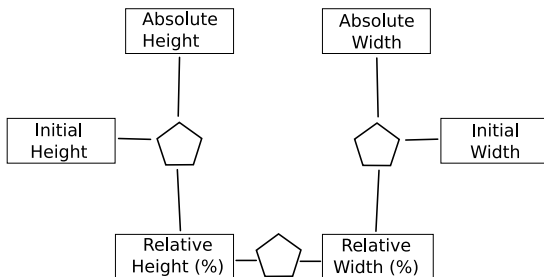
# Core of the Model: Multi-way Dataflow Constraint System

# Core of the Model: Multi-way Dataflow Constraint System



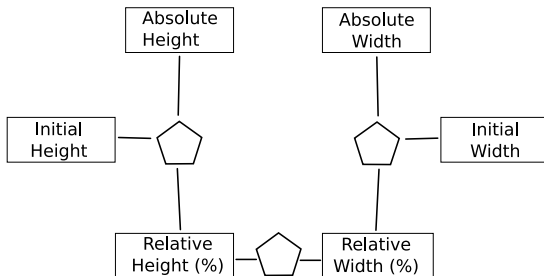
- Variables ...

# Core of the Model: Multi-way Dataflow Constraint System



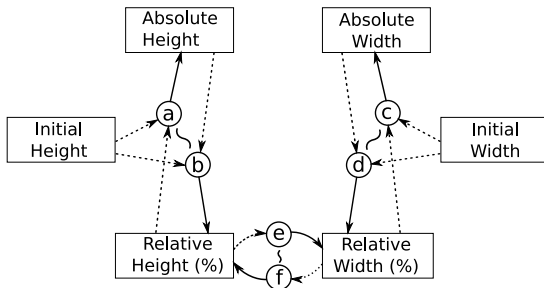
- Variables ...
- tied together by **constraints** ...

# Core of the Model: Multi-way Dataflow Constraint System



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

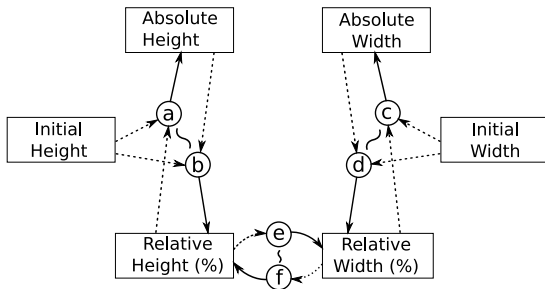
# Core of the Model: Multi-way Dataflow Constraint System



- 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**



# Core of the Model: Multi-way Dataflow Constraint System



- 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:  $\text{absolute\_height} = \text{initial\_height} * \text{relative\_height} / 100;$
  - b:  $\text{relative\_height} = (\text{absolute\_height} / \text{initial\_height}) * 100;$

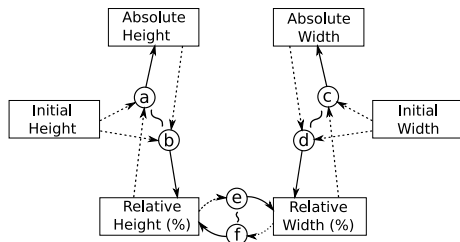
# Multi-way Dataflow Constraint Systems

Resize Image

Initial Height	1500	Initial Width	2100
Absolute Height	1500.0	Absolute Width	2100
Relative Height (%)	100.0	Relative Width (%)	100.0

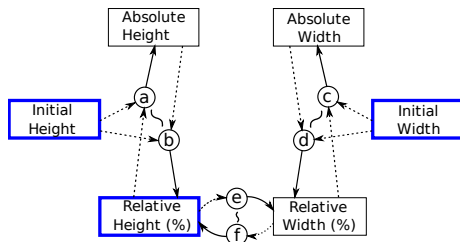
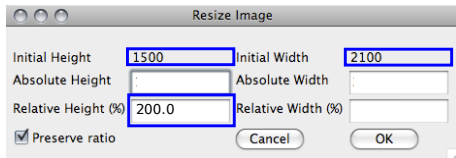
☒ Preserve ratio

Cancel OK



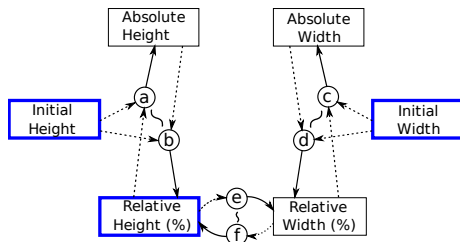
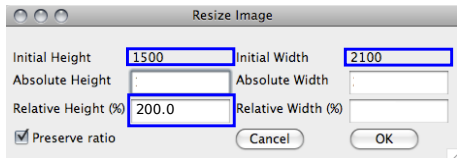
- Restoring consistency is now just solving the system

# Multi-way Dataflow Constraint Systems



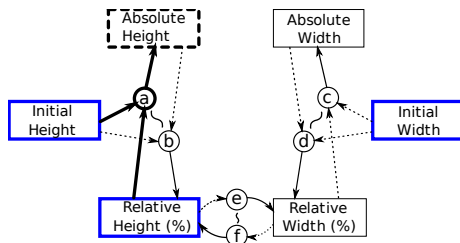
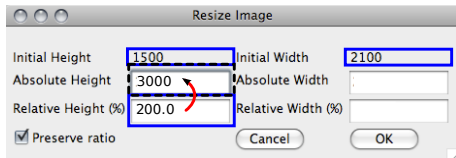
- Restoring consistency is now just solving the system
- Solution defines a **dataflow**

# Multi-way Dataflow Constraint Systems



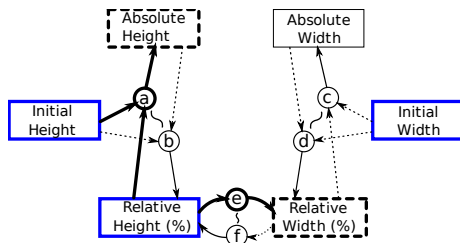
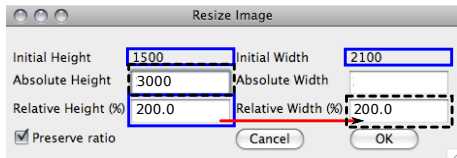
- Restoring consistency is now just solving the system
- Solution defines a **dataflow**
  - Selection of methods (in order) such that
    - all constraints enforced
    - no two methods output to same variable

# Multi-way Dataflow Constraint Systems



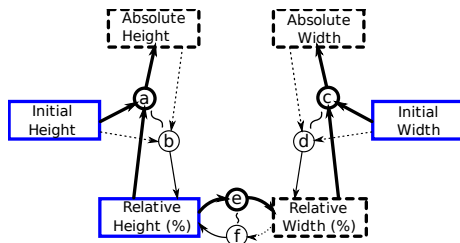
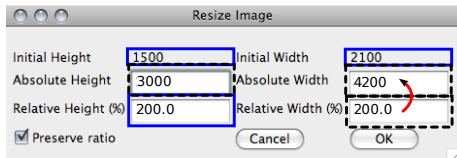
- Restoring consistency is now just solving the system
- Solution defines a **dataflow**
  - Selection of methods (in order) such that
    - all constraints enforced
    - no two methods output to same variable

# Multi-way Dataflow Constraint Systems



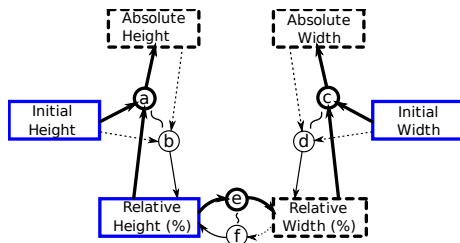
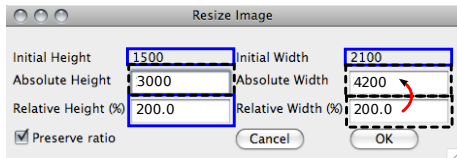
- Restoring consistency is now just solving the system
- Solution defines a **dataflow**
  - Selection of methods (in order) such that
    - all constraints enforced
    - no two methods output to same variable

# Multi-way Dataflow Constraint Systems



- 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

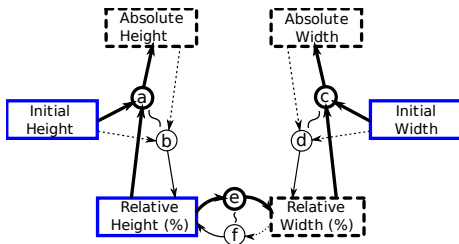
# Multi-way Dataflow Constraint Systems



- 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.  $a, e \rightarrow c$

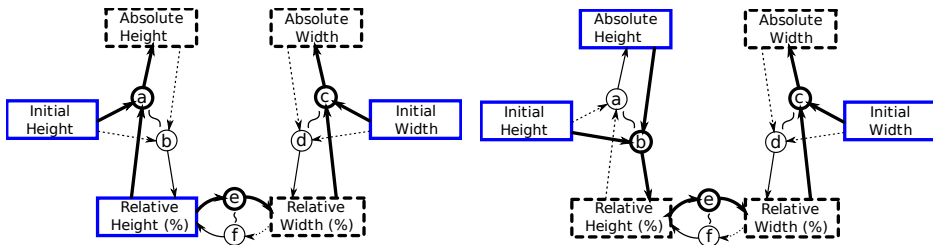


## Picking the “right” solution



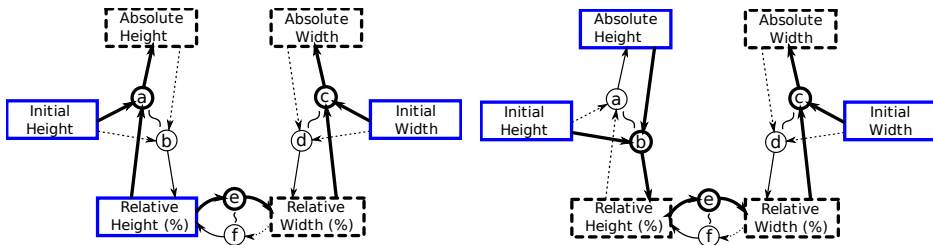
- Programmer only defines relations and their methods, not which method to execute and when  $\Rightarrow$  often multiple solutions

## Picking the “right” solution



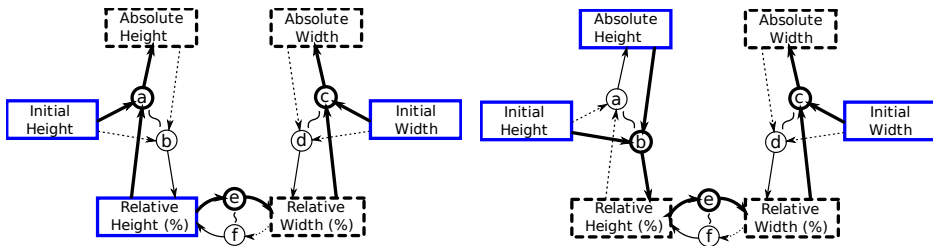
- Programmer only defines relations and their methods, not which method to execute and when  $\Rightarrow$  often multiple solutions

## Picking the “right” solution



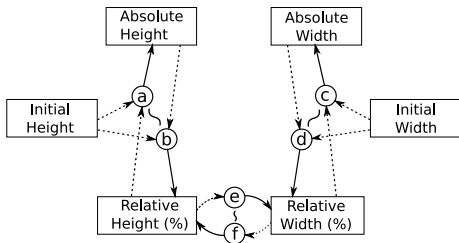
- Programmer only defines relations and their methods, not which method to execute and when  $\Rightarrow$  often multiple solutions
  - Need a way to order them

## Picking the “right” solution



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

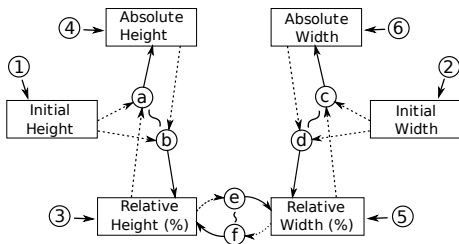
## Picking the “right” solution



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  $\Rightarrow$  often multiple solutions
  - Need a way to order them
- In general, want to prefer methods that change older values
- **Priorities**

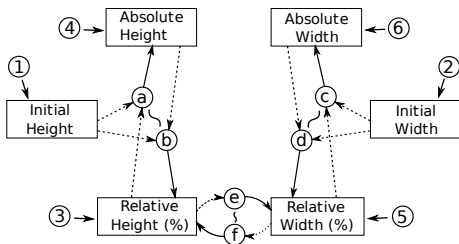
## Picking the “right” solution



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  $\Rightarrow$  often multiple solutions
  - Need a way to order them
- In general, want to prefer methods that change older values
- **Priorities = Hierarchical Stay Constraints**

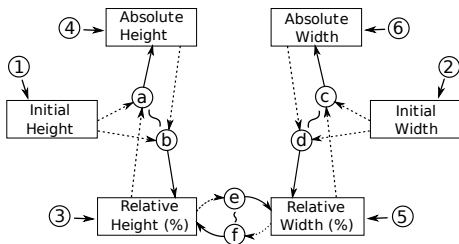
## Picking the “right” solution



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  $\Rightarrow$  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

## Picking the “right” solution



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  $\Rightarrow$  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

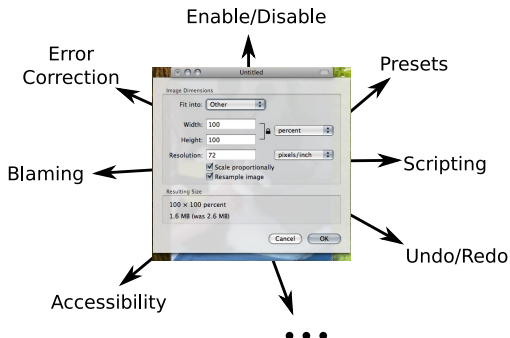


# Explicit Algorithm for Command Parameter Synthesis

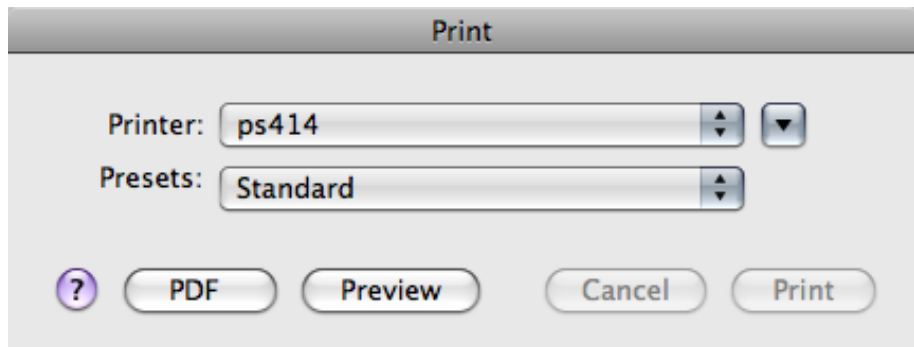
- Each UI element has a variable in a constraint system
- Event handling code becomes auto-generated boilerplate
  - Value modification generates a request to the constraint system to modify one variable and its priority, and solve
  - At all times, the UI element shows the value of the variable in the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

# Property Model

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

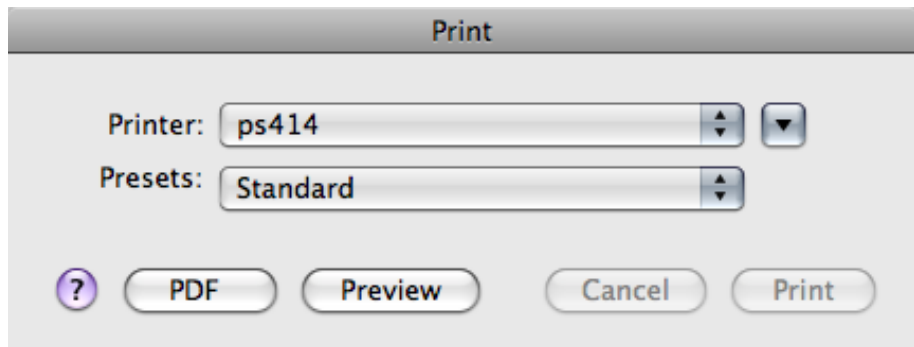


# Enablement



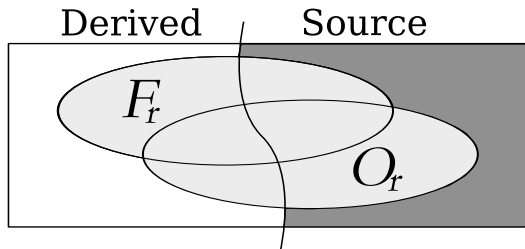
- Currently programmers must explicitly express conditions when a widget should be disabled
  - Use “rules-of-thumb,” usability and user guidelines, give up, ...
- We are able to automate enablement/disablement tracking using property models

# Enablement



- Currently programmers must explicitly express conditions when a widget should be disabled
  - Use “rules-of-thumb,” usability and user guidelines, give up, ...
- We are able to automate enablement/disablement tracking using property models

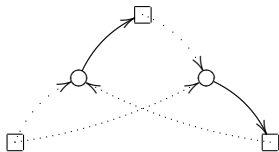
## Enablement: Pictorially



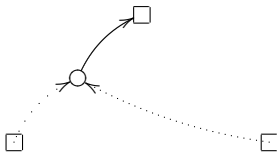
- $O_r$  are all variables relevant to at least one output.
- $F_r$  are all variables relevant to at least one failed invariant.
- $S \setminus (O_r \cup F_r)$  can be safely disabled (dark gray)

# Graphical Overview

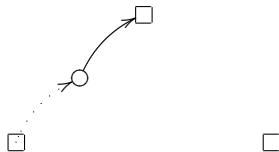
Constraint Graph



Solution Graph



Evaluation Graph



# Current Status and Future Directions

- Early experience deploying property models for command parameter synthesis at Adobe
  - Code reductions of a factor of 8 to 10
  - Fewer defects
  - Consistency among user interfaces
- Opportunities for user interfaces using property models
  - Currently working on model 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 have ideas about applying property models to other kinds of document modeling

## Questions?

