Background: General Research Interests

- Goal: improve software construction
  - Fewer defects
  - Increased productivity
  - Better quality
- Means:
  - Better languages
  - Better libraries
  - Better abstractions
  - Better tools
  - Reuse, Generic Programming
User interfaces are a costly, error-prone area of software construction
- Coding effort
- Defects
- Non-uniform behavior
- Costly to users: lost time, frustration, errors
User interfaces are a costly, error-prone area of software construction

- Coding effort
- Defects
- Non-uniform behavior
- Costly to users: lost time, frustration, errors
Why should we care?


*User frustration in the use of information and computing technology is a pervasive and persistent problem. When computers crash, network congestion causes delays, and poor user interfaces trigger confusion there are dramatic consequences for individuals, organizations, and society. These frustrations not only cause personal dissatisfaction and loss of self-efficacy, but may disrupt workplaces, slow learning, and reduce participation in local and national communities.*
What can we do to improve the situation?
Non-Scientific Observations about Economics of UI development

- Some applications have millions of users.
Non-Scientific Observations about Economics of UI development

- Some applications have millions of users.
  - They tend to have nice, intuitive, feature rich, smooth, and defect free UIs.
Non-Scientific Observations about Economics of UI development

- Some applications have millions of users.
  - They tend to have nice, intuitive, feature rich, smooth, and defect free UIs
  - Think of Google, Orbitz, Adobe Photoshop, ...

Some applications don’t have millions of users

They tend to not have nice, intuitive, feature rich, smooth, and defect free UIs

Think of your favourite in-house application, mom-and-pop website

We spend a significant amount of our time with the latter kind of systems

The economically viable way to get nice, intuitive, feature rich, smooth, and defect free UIs to the latter family of applications is to make those traits cost very little (preferably nothing).
Non-Scientific Observations about Economics of UI development

- Some applications have millions of users.
  - They tend to have nice, intuitive, feature rich, smooth, and defect free UIs
  - Think of Google, Orbitz, Adobe Photoshop, ...
- Some applications don’t have millions of users
Non-Scientific Observations about Economics of UI development

- Some applications have millions of users.
  - They tend to have nice, intuitive, feature rich, smooth, and defect free UIs
  - Think of Google, Orbitz, Adobe Photoshop, ...
- Some applications don’t have millions of users
  - They tend to not have nice, intuitive, feature rich, smooth, and defect free UIs
Some applications have millions of users. They tend to have nice, intuitive, feature rich, smooth, and defect free UIs. Think of Google, Orbitz, Adobe Photoshop, ...

Some applications don’t have millions of users. They tend to not have nice, intuitive, feature rich, smooth, and defect free UIs. Think of your favourite in-house application, mom-and-pop website.
Non-Scientific Observations about Economics of UI development

- Some applications have millions of users.
  - They tend to have nice, intuitive, feature rich, smooth, and defect free UIs
  - Think of Google, Orbitz, Adobe Photoshop, ...

- Some applications don’t have millions of users
  - They tend to not have nice, intuitive, feature rich, smooth, and defect free UIs
  - Think of your favourite in-house application, mom-and-pop website

- We spend a significant amount of our time with the latter kind of systems
Non-Scientific Observations about Economics of UI development

- Some applications have millions of users.
  - They tend to have nice, intuitive, feature rich, smooth, and defect free UIs
  - Think of Google, Orbitz, Adobe Photoshop, ...

- Some applications don’t have millions of users
  - They tend to not have nice, intuitive, feature rich, smooth, and defect free UIs
  - Think of your favourite in-house application, mom-and-pop website

- We spend a significant amount of our time with the latter kind of systems
  
  The economically viable way to get nice, intuitive, feature rich, smooth, and defect free UIs to the latter family of applications is to make those traits cost very little (preferably nothing).
Increase reuse

- Very little reuse in user interfaces
  - Compositions of interface components are not reusable
  - Complex interface behaviors are not reusable
  - Each user interface is a custom, unique solution
- We are working to identify algorithms that capture user interface behavior
- Practical goal: library of reusable user interface functionalities
Functional Dependencies in a User Interface

![Save Image As... dialog box]

- **File name:** picture.bmp
- **Save as type:** Raw bitmap (.bmp)
- **Compression ratio:**
- **Image quality:**

[Save] [Cancel]
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 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 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
    width = float(self.Controls["AbsolutePx"]["Width"].GetValue())
    pct = width / 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))

    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))

    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)))

if __name__ == '__main__':
    def ChangeConstrainState(self, event):
        constrained = self.Controls["Constrain"].GetValue()
        if constrained: # update height & height%
            self.Controls["Relative%"]["Height"].SetValue(str(height))
            self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

            # update width & width%
            self.Controls["Relative%"]["Width"].SetValue(str(width))
            self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))

if __name__ == '__main__':
    # initialize Resize Image window
    ResizeImage()
    # set initial sizes
    InitialSize = [1500, 2100]
    # set initial values
    InitialValue = [1500.0, 2100.0]
    # set percent values
    RelativeValue = [100.0, 100.0]

    # set controls
    Controls = [...

    # set event handlers
    eventHandlerAbsH = EventHandler()
    eventHandlerAbsW = EventHandler()
    eventHandlerRelH = EventHandler()
    eventHandlerRelW = EventHandler()

    # set last updated
    LastUpdated = "Height"

    # set preserve ratio
    if constrained:
        if LastUpdated == "Height": # update width px & %
            width = height * self.InitialSize[self.Width] / 100
            self.Controls["AbsolutePx"]["Width"].SetValue(str(round(width)))
        else: # update width px & %
            height = width * self.InitialSize[self.Height] / 100
            self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

    # set event handlers
    eventHandlerAbsH = EventHandler()
    eventHandlerAbsW = EventHandler()
    eventHandlerRelH = EventHandler()
    eventHandlerRelW = EventHandler()

    # set last updated
    LastUpdated = "Height"

    # set preserve ratio
    if constrained: # update height & height%
        self.Controls["Relative%"]["Height"].SetValue(str(height))
        self.Controls["AbsolutePx"]["Height"].SetValue(str(round(height)))

        # update width & width%
        self.Controls["Relative%"]["Width"].SetValue(str(width))
        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))
                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)))

    ...
Property Models

- Foundation for defining generic, reusable user interface behavior
- Explicitly models functional dependencies among interface components
- Hierarchical multi-way dataflow constraint system
Dependency Graphs : Save Image Dialog

sheet save_image_file {

    interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

    logic:
    relate {
        compression_ratio <= 100 - 4 * (100 - image_quality);
        image_quality <= 100 - (100 - compression_ratio) / 4;
    }

    output:
    result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }
    { type: file_type, name: file_name };

    invariant: check_name <= file_name != "";
}
Dependency Graphs : Save Image Dialog

sheet save_image_file {
    interface:
        file_name : "";
        file_type : "bmp";
        compression_ratio : 100;
        image_quality : 100;

    logic:
        relate {
            compression_ratio \leq 100 - 4 \times (100 - \text{image_quality});
            image_quality \leq 100 - (100 - \text{compression_ratio}) / 4;
        }

    output:
        result \leq (\text{file_type} == "jpeg") ?
            { type: file_type, name: file_name, ratio: compression_ratio };
            { type: file_type, name: file_name };

    invariant: check_name \leq file_name != "";
}
Dependency Graphs : Save Image Dialog

```
sheet save_image_file {

    interface:
        file_name : "";
        file_type : "bmp";
        compression_ratio : 100;
        image_quality : 100;

    logic:
        relate {
            compression_ratio <= 100 - 4 * (100 - image_quality);
            image_quality <= 100 - (100 - compression_ratio) / 4;
        }

    output:
        result <= (file_type == "jpeg") ?
            { type: file_type, name: file_name, ratio: compression_ratio }:
            { type: file_type, name: file_name };

    invariant: check_name <= file_name != "";
}
```
Dependency Graphs : Save Image Dialog

```
sheet save_image_file {

  interface:
  file_name : "";
  file_type : "bmp";
  compression_ratio : 100;
  image_quality : 100;

  logic:
  relate {
    compression_ratio <= 100 - 4 * (100 - image_quality);
    image_quality <= 100 - (100 - compression_ratio) / 4;
  }

  output:
  result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }:
    { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
```
sheet save_image_file {
  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio <= 100 − 4 * (100 − image_quality);
      image_quality <= 100 − (100 − compression_ratio) / 4;
    }

  output:
    result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }:
      { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
Dependency Graphs : Save Image Dialog

```
sheet save_image_file {

  interface:
  file_name : "";
  file_type : "bmp";
  compression_ratio : 100;
  image_quality : 100;

  logic:
  relate {
    compression_ratio <= 100 - 4 * (100 - image_quality);
    image_quality <= 100 - (100 - compression_ratio) / 4;
  }

  output:
  result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }:
    { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
```
Dependency Graphs: Save Image Dialog

```plaintext
sheet save_image_file {

  interface:
  file_name : "";
  file_type : "bmp";
  compression_ratio : 100;
  image_quality : 100;

  logic:
  relate {
    compression_ratio <= 100 - 4 * (100 - image_quality);
    image_quality <= 100 - (100 - compression_ratio) / 4;
  }

  output:
  result <= (file_type == "jpeg") ?
  { type: file_type, name: file_name, ratio: compression_ratio }
  : { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
```
Dependency Graphs: Save Image Dialog

```
sheet save_image_file {

  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }

  output:
    result <= (file_type == "jpeg") ?
      { type: file_type, name: file_name, ratio: compression_ratio }
    { type: file_type, name: file_name }

  invariant: check_name <= file_name != "";
}
```
Dependency Graphs : Save Image Dialog

```
sheet save_image_file {
  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;
  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }
  output:
    result <= (file_type == "jpeg") ?
      {
        type: file_type, name: file_name, ratio: compression_ratio
      };
      {
        type: file_type, name: file_name
      };
  invariant: check_name <= file_name != "";
}
```
Dependency Graphs : Save Image Dialog

sheet save_image_file {

interface:
  file_name : "";
  file_type : "bmp";
  compression_ratio : 100;
  image_quality : 100;

logic:
  relate {
    compression_ratio <= 100 - 4 * (100 - image_quality);
    image_quality <= 100 - (100 - compression_ratio) / 4;
  }

output:
  result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio } :
    { type: file_type, name: file_name };

invariant: check_name <= file_name != "";
}
sheet save_image_file {

    interface:
    file_name := "";
    file_type := "bmp";
    compression_ratio := 100;
    image_quality := 100;

    logic:
    relate {
        compression_ratio <= 100 − 4 * (100 − image_quality);
        image_quality <= 100 − (100 − compression_ratio) / 4;
    }

    output:
    result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }:
    { type: file_type, name: file_name };

    invariant: check_name <= file_name != "";
}

Dependency Graphs : Save Image Dialog

```
sheet save_image_file {
    interface:
        file_name : "";
        file_type : "bmp"
    file
    compression_ratio : 100;
    image_quality : 100;

    logic:
        relate {
            compression_ratio <= 100 - 4 * (100 - image_quality);
            image_quality <= 100 - (100 - compression_ratio) / 4;
        }

    output:
        result <= (file_type == "jpeg") ?
            { type: file_type, name: file_name, ratio: compression_ratio }
            { type: file_type, name: file_name };

    invariant: check_name <= file_name != "";
}
```
Dependency Graphs : Save Image Dialog

sheet save_image_file {
  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }

  output:
    result <= (file_type == "jpeg") ?
      { type: file_type, name: file_name, ratio: compression_ratio }
      { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
Dependency Graphs : Save Image Dialog

sheet save_image_file {
    interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

    logic:
    relate {
        compression_ratio <= 100 - 4 * (100 - image_quality);
        image_quality <= 100 - (100 - compression_ratio) / 4;
    }

    output:
    result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }
    : { type: file_type, name: file_name };

    invariant: check_name <= file_name != "";
}
Dependency Graphs : Save Image Dialog

```
sheet save_image_file {

  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }

  output:
    result <= (file_type == "jpeg") ?
      { type: file_type, name: file_name, ratio: compression_ratio }:
      { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
```
Dependency Graphs : Save Image Dialog

sheet save_image_file {
  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }

  output:
    result <= (file_type == "jpeg")?
      { type: file_type, name: file_name, ratio: compression_ratio }:
      { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
sheet save_image_file {
  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;
  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }
  output:
    result <= (file.type == "jpeg") ?
      { type: file.type, name: file_name, ratio: compression_ratio }
      { type: file.type, name: file_name };
  invariant: check_name <= file_name != "";
}
Dependency Graphs : Save Image Dialog

sheet save_image_file {
    interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

    logic:
    relate {
        compression_ratio <= 100 − 4 * (100 − image_quality);
        image_quality <= 100 − (100 − compression_ratio) / 4;
    }

    output:
    result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }:
    { type: file_type, name: file_name };

    invariant: check_name <= file_name != "";
}
### Dependency Graphs: Save Image Dialog

**Sheet** `save_image_file`

**Interface:**
- `file_name` : "";
- `file_type` : "bmp";
- `compression_ratio` : 100;
- `image_quality` : 100;

**Logic:**

```
relate {
    compression_ratio <= 100 - 4 * (100 - image_quality);
    image_quality <= 100 - (100 - compression_ratio) / 4;
}
```

**Output:**

```
result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio };
    { type: file_type, name: file_name };}
```

**Invariant:**

```
check_name <= file_name != "";
```
sheet save_image_file {

interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

logic:
    relate {
        compression_ratio <= 100 - 4 * (100 - image_quality);
        image_quality <= 100 - (100 - compression_ratio) / 4;
    }

output:
    result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }:
    { type: file_type, name: file_name };

invariant: check_name <= file_name != "";
}
Dependency Graphs: Save Image Dialog

```
sheet save_image_file {
  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }

  output:
    result <= (file_type == "jpeg") ?
      { type: file_type, name: file_name, ratio: compression_ratio } :
      { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
```
Dependency Graphs : Save Image Dialog

sheet save_image_file {
  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }

  output:
    result <= (file_type == "jpeg") ? {
      type: file_type, name: file_name, ratio: compression_ratio
    }:
      {
        type: file_type, name: file_name
      };

  invariant: check_name <= file_name != "";
}
Dependency Graphs : Save Image Dialog

sheet save_image_file {
    interface:
        file_name : "";
        file_type : "bmp";
        compression_ratio : 100;
        image_quality : 100;

    logic:
        relate {
            compression_ratio <= 100 - 4 * (100 - image_quality);
            image_quality <= 100 - (100 - compression_ratio) / 4;
        }

    output:
        result <= (file_type == "jpeg") ?
            { type: file_type, name: file_name, ratio: compression_ratio }:
            { type: file_type, name: file_name };

    invariant: check_name <= file_name != "";
}

Dependency Graphs : Save Image Dialog

sheet save_image_file {

    interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

    logic:
    relate {
        compression_ratio <= 100 - 4 * (100 - image_quality);
        image_quality <= 100 - (100 - compression_ratio) / 4;
    }

    output:
    result <= (file_type == "jpeg") ?
    { type: file_type, name: file_name, ratio: compression_ratio }
    { type: file_type, name: file_name };

    invariant: check_name <= file_name != ""

Solution:

- Exactly one method is selected from each constraint
Dependency Graphs : Save Image Dialog

sheet save_image_file {

  interface:
  file_name : "";
  file_type : "bmp";
  compression_ratio : 100;
  image_quality : 100;

  logic:
  relate {
    compression_ratio <= 100 - 4 * (100 - image_quality);
    image_quality <= 100 - (100 - compression_ratio) / 4;
  }

  output:
  result <= (file_type == "jpeg") ?
  { type: file_type, name: file_name, ratio: compression_ratio }:
  { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}

Solution:
- Exactly one method is selected from each constraint
- No variable is output by more than one method
Dependency Graphs: Save Image Dialog

**Solution:**

- Exactly one method is selected from each constraint
- No variable is output by more than one method

```plaintext
sheet save_image_file {
  interface:
  file_name : "";
  file_type : "bmp";
  compression_ratio : 100;
  image_quality : 100;

  logic:
  relate {
    compression_ratio <= 100 - 4 * (100 - image_quality);
    image_quality <= 100 - (100 - compression_ratio) / 4;
  }

  output:
  result <= (file_type == "jpeg") ?
  { type: file_type, name: file_name, ratio: compression_ratio };
  { type: file_type, name: file_name };

  invariant: check_name <= file_name != "";
}
```
Dependency Graphs: Save Image Dialog

sheet save_image_file {
  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio <= 100 - 4 * (100 - image_quality);
      image_quality <= 100 - (100 - compression_ratio) / 4;
    }

  output:
    result <= (file_type == "jpeg") ?
      { type: file_type, name: file_name, ratio: compression_ratio }
    { type: file_type, name: file_name }

  invariant: check_name <= file_name != "";
}
Property Models

- Represented by three different graphs
  - Constraint graph: all *possible* dependencies in *any* state of dialog
  - Solution graph: all *possible* dependencies for a particular editing history with any values of variables
  - Evaluation graph: all *actual* dependencies for a particular editing history with particular values of variables
Property Models

- Capture functional dependencies explicitly in a concrete data structure
- Aware of different “types” of interface components, e.g. outputs, invariants
- Enable implementation of user interface behaviors as generic, reusable algorithms
Enablement

- Each platform provides different guidelines for implementing enablement
- Two kinds of enablement
  - **Widget enablement**  A widget bound to a value that cannot currently affect the end result of the dialog should be disabled
  - **Command activation**  A widget that triggers a command with a currently unsatisfied precondition should be deactivated
Widget Enablement: Save Image Dialog

- File name: picture.bmp
- Save as type: Raw bitmap (.bmp)
- Compression ratio:
- Image quality:

[Save] [Cancel]
A widget bound to a value that cannot currently affect the end result of the dialog should be disabled.
Widget Enablement : Save Image Dialog
A value of a variable $v$ cannot affect an output variable if there exists no such variable $w$ that is (1) an ancestor of $v$ in the solution graph, and (2) reachable from $v$ in the constraint graph, and (3) reaches an output variable in the evaluation graph.
Command Activation: Save Image Dialog
A widget that triggers a command with a currently unsatisfied precondition should be deactivated.
Command Activation : Save Image Dialog
An output variable $o$ should be deactivated if, in the evaluation graph, there exists a variable $w$ such that (1) $w$ reaches $o$, and (2) $w$ reaches a failed invariant.
Widget enablement code is in red, command activation code in blue.

sheet save_image_file {

  interface:
    file_name : "";
    file_type : "bmp";
    compression_ratio : 100;
    image_quality : 100;

  logic:
    relate {
      compression_ratio ⇐ 100 − 4 * (100 − image_quality);
      image_quality ⇐ 100 − (100 − compression_ratio) / 4;
    }

  output:
    result ⇐ (file_type == "jpeg") ?
      { type: file_type, name: file_name, ratio: compression_ratio } :
      { type: file_type, name: file_name };

  invariant:
    check_name ⇐ file_name != "";
}

layout save_image_file {

  view dialog(name: "Save Image As...",
    placement: place_column, spacing: 6,
    child_horizontal: align_fill)
  {
    edit_text(bind: @file_name, name: "File name:");
    popup(bind: @file_type, name: "Save as type:", items: [
      { name: "Raw bitmap (.bmp)", value: "bmp" },
      { name: "JPEG (.jpeg)", value: "jpeg" }
    ]);}

  row() {
    column(child_horizontal: align_right) {
      label(name: "Compression ratio:");
      label(name: "Image quality:");
    }
    column(horizontal: align_fill, child_horizontal: align_fill) {
      slider(bind: @compression_ratio,
        format: { first: 1, last: 100, interval: 1 } );
      slider(bind: @image_quality,
        format: { first: 1, last: 100, interval: 1 } );
    }
  }

  row(horizontal: align_right) {
    button(name: "Save", action: @ok, bind: @result, default: true);
    button(name: "Cancel", action: @cancel);
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 a property model library, 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

![Graph showing the number of bugs reported over reporting weeks for different categories: AE1, AE2, AE3, and TF. The x-axis represents the reporting week, and the y-axis represents the number of bugs reported.]
Related work

- Data-flow constraints for user interfaces studied extensively (since 70’s)
  - Sketchpad, Amulet, Garnet, ThingLab I and II, DeltaBlue, SkyBlue, OpenLaszlo, ...
- Constraints mainly used for layout where one-way constraints are rather standard e.g., in many diagram drawing tools
- Retrospect by Amulet group concludes that it is unlikely that constraint systems will ever be used for much other than layout
- Amulet and the related systems integrate a constraint solver into a general purpose programming language. Variable valuations based on user-defined constraints maintained, but the state of constraint system (which functional dependencies are in effect) is not manifestly observable
Related work

- Data-flow constraints for user interfaces studied extensively (since 70’s)
  - Sketchpad, Amulet, Garnet, ThingLab I and II, DeltaBlue, SkyBlue, OpenLaszlo, ...

- Constraints mainly used for layout where one-way constraints are rather standard e.g., in many diagram drawing tools

- Retrospect by Amulet group concludes that it is unlikely that constraint systems will ever be used for much other than layout

- Amulet and the related systems integrate a constraint solver into a general purpose programming language. Variable valuations based on user-defined constraints maintained, but the state of constraint system (which functional dependencies are in effect) is not manifestly observable
Related work

- Data-flow constraints for user interfaces studied extensively (since 70’s)
  - Sketchpad, Amulet, Garnet, ThingLab I and II, DeltaBlue, SkyBlue, OpenLaszlo, ...
- Constraints mainly used for layout where one-way constraints are rather standard e.g., in many diagram drawing tools
- Retrospect by Amulet group concludes that it is unlikely that constraint systems will ever be used for much other than layout
- Amulet and the related systems integrate a constraint solver into a general purpose programming language. Variable valuations based on user-defined constraints maintained, but the state of constraint system (which functional dependencies are in effect) is not manifestly observable
Demo

http://leela.cs.tamu.edu/pm/index.php
http://parasol.tamu.edu/groups/pttlgroup/property-models/pmeval-flapjax/index.php