Helping Programmers Help Users

John Freeman\textsuperscript{1}  Jaakko Järvi\textsuperscript{1}  Wonseok Kim\textsuperscript{1}
Mat Marcus\textsuperscript{2}  Sean Parent\textsuperscript{3}

Texas A&M University\textsuperscript{1}
Canyonlands Software Design\textsuperscript{2}
Adobe Systems, Inc.\textsuperscript{3}
The Idea

The problem:
- Much frustration with existing software
- Help features are uncommon

Our idea:
- Make it easier to include help features
- Encapsulate features into reusable components
This Talk

- **Background**
  - Constraint systems
  - Property models

- **Reusable behaviors**
  - Visualizing dataflow
  - Controlling dataflow (pinning)
  - Explaining command availability
Constraint Graphs

- One variable for each value
- One constraint for each relationship among values
Constraint Graphs

- Methods enforce constraints

\[ \text{Ratio} := f(\text{Quality}) \]
\[ \text{Quality} := g(\text{Ratio}) \]
Constraint Graphs

- Output variables to aggregate command parameters
- Constraint graph
Constraint Graphs

- Whenever value changes, solve and evaluate
- Solution := one method for every constraint s.t. no variable written twice
- Multiple solutions may be possible
Constraint Graphs

Whenever value changes, solve and evaluate
Solution := one method for every constraint s.t. no variable written twice
Multiple solutions may be possible
Constraint Graphs

- Use editing history to order solutions
  - Try to preserve more recently edited values
- Each variable gets stay constraint with priority (strength)
- When solving, weaker constraints can be dropped in favor of stronger ones
Constraint Graphs

- Use editing history to order solutions
  - Try to preserve more recently edited values
- Each variable gets stay constraint with priority (strength)
- When solving, weaker constraints can be dropped in favor of stronger ones

Solution graph
Constraint Graphs

- Evaluate := execute each method in topographical order
- Some methods may not use all of their inputs
Constraint Graphs

Evaluate := execute each method in topographical order

Some methods may not use all of their inputs
Constraint Graphs

- Evaluate := execute each method in topographical order
- Some methods may not use all of their inputs
- Evaluation graph
The Three Graphs

- **Constraint Graph**
  - File name
  - File type
  - Ratio
  - Quality

- **Solution Graph**
  - File name
  - File type
  - Ratio
  - Quality

- **Evaluation Graph**
  - File name
  - File type
  - Ratio
  - Quality

Each graph represents the dependencies for given sets of editing histories and valuations.
The Three Graphs

- **Constraint Graph**
  - File name
  - File type
  - Ratio
  - Quality

- **Solution Graph**
  - File name
  - File type
  - Ratio
  - Quality

- **Evaluation Graph**
  - File name
  - File type
  - Ratio
  - Quality

- Each graph represents the dependencies for given sets of editing histories and valuations

<table>
<thead>
<tr>
<th>Graph</th>
<th>History</th>
<th>Valuation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Solution</td>
<td>One</td>
<td>All</td>
</tr>
<tr>
<td>Evaluation</td>
<td>One</td>
<td>One</td>
</tr>
</tbody>
</table>
Users may not understand relationships among values

- Drawing them helps close gulf of evaluation
- Property model allows feature to be packaged in reusable component
Visualizing Dataflow

- Users may not understand relationships among values
  - Drawing them helps close gulf of evaluation
- Property model allows feature to be packaged in reusable component
Controlling Dataflow (Pinning)
Controlling Dataflow (Pinning)

- Users may not be able to predict dataflow after their next change
- Users may not want the dataflow that the UI picks
Pinning lets users protect values from UI changes
Stay constraint gets unbeatable strength
Virtually removes methods from constraint graph
If too many variables pinned, solution impossible
The system detects this situation and prevents it
Controlling Dataflow (Pinning)

- When unpinned, variable gets highest priority
Explaining Command Availability

- Programmer specifies command preconditions, e.g.
  \[ \text{nights} \geq 1 \]

- System blames variables responsible for unsatisfied preconditions
- If blamed variable reaches output variable in evaluation graph, 
  - Associated command is disabled
Summary

- UIs can still customize the visual aspects of these features
- These behaviors seem simple
- Explicit dependency information makes them simple
- Implementing these behaviors by hand for each interface is tedious and error-prone
- Property models enable reusable implementation
- By lowering cost of help features,
  - Encourage their adoption
  - Reduce user frustration