mylar

a degree-of-interest model for eclipse

Mik Kersten
Gail Murphy
Shawn Minto

University of British Columbia

EclipseCon 2005
public class MultiPageEditor extends MultiPageEditorPart

/** The text editor used in page 0. */
private TextEditor editor;

/** The font chosen in page 1. */
private Font font;

/** The text widget used in page 2. */
private StyledText text;

/**
 * Creates a multi-page editor example.
 */

public MultiPageEditor()
{
    super();
    ResourcesPlugin.getWorkspace().addResourceChangeListener(

/**
 * Creates page 0 of the multi-page editor,
 * which contains a text editor.
 */

void createPage0()
{
    List list = new ArrayList();
    for (Iterator iter = list.iterator(); iter.hasNext();)
        Object element = (Object) iter.next();
}
protected boolean getFlag(int flag) {
    return (flags & flag) != 0;
}

/**
 * @see IFigure#getFont()
 */

public Font getFont() {
    if (font != null)
        return font;
    if (getParent() != null)
        return getParent().getFont();
    return null;
}
Mylar¹
shows you only what you’re working on

1a. Aluminized film used to avoid blindness when staring at a solar eclipse
1b. UI ‘skin’ used to avoid information blindness when staring at Eclipse
<table>
<thead>
<tr>
<th>Mylar</th>
</tr>
</thead>
<tbody>
<tr>
<td>• How does it present what’s interesting?</td>
</tr>
<tr>
<td>• How does it determine what’s interesting?</td>
</tr>
<tr>
<td>• How does it support your work practice?</td>
</tr>
</tbody>
</table>
```java
protected boolean getFlag(int flag) {

    /**
     * @return
     */
    public Font getFont() {
        if (font != null)
            return font;
        if (getParent() != null)
            return getParent().getFont();
        return null;
    }

    /**
     * @return
     */
    public Insets getInsets() {

    }

    /**
     * @return
     */
    public LayoutManager getLayoutManager() {

    }

    /**
     * @return
     */
    protected Iterator getListeners(Class clazz) {

    }

    /**
     * @return
     */
    public Color getLocalBackgroundColor() {

    }

    /**
     * @return
     */
    protected Font getLocalFont() {

    }
```
Mylar

- How does it present what’s interesting?
  - Bolding, automatic expansion and highlighting
  - Collapses and filters uninteresting elements

- How does it determine what’s interesting?

- How does it support your work practice?
Demo 1

- Setup
  - Bug: 32618
  - Search: bugzilla, java refs, xml
  - Auto filtered
- Look at ClassEditPart
- Navigate to Font constructors
- Navigate to Label
- Toggle outline, editor folding
- Show declarations on ClassEditPart
Mylar

• How does it present what’s interesting?
  – Bolding, automatic expansion and highlighting
  – Collapses and filters uninteresting elements

• How does it determine what’s interesting?
  – Monitors editing and navigation
  – Encodes degree of interest, periodic decay

• How does it support your work practice?
Demo 2

- Show task list (been working on a bug report)
- Deactivate, activate
- Discuss a new bug report came in, looks related to current task
- Create new bugzilla report: 77941, activate
- Navigate to FigureUtilities set/get Font
- Toggle outline to show highlighter
- Re-activate previous
Mylar

• How does it present what’s interesting?
  – Bolding, automatic expansion and highlighting
  – Collapses and filters uninteresting elements

• How does it determine what’s interesting?
  – Monitors editing and navigation
  – Encodes degree of interest, periodic decay

• How does it support your work practice?
  – Context is explicit, can be saved and restored
  – Multiple tasks can be active
  – Tasks can be linked to bug reports
Mylar

• How does it present what’s interesting?
  – Bolding, automatic expansion and highlighting
  – Collapses and filters uninteresting elements

• How does it determine what’s interesting?
  – Monitors editing and navigation
  – Encodes degree of interest, periodic decay
  – Related elements can also be interesting

• How does it support your work practice?
  – Context is explicit, can be saved and restored
  – Multiple tasks can be active
  – Tasks can be linked to bug reports
Demo 3

- Browse to Label and add 3rd arg to constructor
- Mention that errors only have transient interest
- Discuss no more need of errors view
What about search?

- **Eclipse**
  - Invoke manually and inspect long list of results

- **Mylar**
  - Landmarks are the most interesting elements
  - *Active Search* updates related elements eagerly in the background
  - Results show interest level

- **Related elements**
  - References, implementors
  - Read, write access
  - Unit tests
  - Bugzilla & XML references
Type Hierarchy

- **Eclipse**
  - Invite the hierarchy on an element, then wait

- **Mylar**
  - *Active Type Hierarchy* of landmarks is updated in the background
  - Inheritance context of what you’re working on is always visible

- **Demo…**
Demo 4

- Show active search view
- Toggle Java and other refs
- Open EDiagramEditor and make it a landmark
  - Make landmarks clear
- Navigate to XML
What about search scope?

- **Degree of separation**
  - Distance of related elements from the current task context

- E.g. for Java elements degrees are:
  1. Landmarks
  2. Interesting elements
  3. Interesting projects
  4. Project dependencies
  5. Workspace

- Demo...
Demo 5

- **Degree of separation**
  - Discuss starting context vs. rich context
  - Note that there are more bug reports
  - More java elements
  - Ranking

- **Bugzilla matches**
  - Separation is consistent for bug reports, first looks at only interesting reports, then repository

- **Scaling factor**
  - I never touched a scrollbar in this demo…
  - First make it flash, then do slow and explain
There’s more...

- Presentation is configurable
  - Highlighters, relative interest, intersection
  - Landmark decorators
  - In-place active views

- AspectJ support

- Task management
  - Planning game tasks
  - Offline editing of bug reports

- Seamless integration
  - Don’t lose anything: filters, sorters
  - Adopt incrementally: pick and choose the views
plug-in details
and project plans
Under the hood

- Interest updated as you work
  - Periodic decay, active search in background
  - Uninteresting files close automatically

- Built on existing models
  - Reuses Java, XML, and Bugzilla models and search facilities
  - Inherently lightweight and lazy structure

- Performance characteristics
  - Reduces memory footprint by limiting view population
Development experiences

• Good modularity in JDT made this possible
  – Filters, sorters, content providers, label providers
  – Able to subclass existing views and Java editor
  – Drive folding, extend search
  – Quality, consistency of the Java model was key
  – This level of integration would have been difficult or impossible in any other IDE

• But..
  – Extending internal APIs
  – Overriding existing providers on initialization
  – Not able to extend Outline view
  – Had to create own in-place view, XML folding
Around the corner

- Collaborative support
  - Sharing task context via Bugzilla attachments
  - Awareness of team members’ task context

- Applies to other views as well
  - Debugger thread tree
  - Visualizations
  - Help documentation
<table>
<thead>
<tr>
<th>Project status</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Mylar is a research project</td>
</tr>
<tr>
<td>• v0.1: august ‘04 user study</td>
</tr>
<tr>
<td>• v0.2: now</td>
</tr>
<tr>
<td>• v0.3: june ’05 user study</td>
</tr>
<tr>
<td>• v0.5: first public release</td>
</tr>
</tbody>
</table>
Summary

- Mylar is two things
  - *Degree of interest*: inspired by Stu Card’s tree viz
  - *Degree of separation*: inspired by Kevin Bacon

- Makes working with very large projects as easy as it is to work with small projects

- Credits
  - Mylar: Mik Kersten and Gail Murphy
  - XML and Bugzilla integration: Shawn Minto
  - Bugzilla plug-in support: Eric Booth, Ken Sueda
  - Funded by IBM CAS, NSERC, UBC

- [www.cs.ubc.ca/~mylar](http://www.cs.ubc.ca/~mylar) questions: beatmik@acm.org