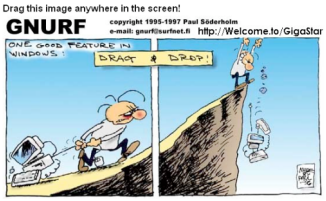

Drag and Drop programming

Cristian Bogdan
cristi@kth.se

DnD: Drag programming

- Drag gesture identification
  - Typically a mouse press, followed by a move (drag)
  - Some option keys may be pressed to change the action
- "Drop source" component (widget)
- Drag cursor/icon
- Drag and drop cancelling gesture (typically Esc key)
  - Current drop destination must find out so it stops drawing the drop response

DnD: Drop programming

- Location-sensitive response at "drop target"
  - "drag under"
  - Not the entire area can be dropped on
  - Drop may be possible but not precisely at the mouse location
    - Text fields show a caret at the drop point
  - Autoscroll to allow dropping on/near list elements not initially visible
  - Open elements of hierarchical structures like trees, to allow drop over them (sometimes in a new window, see Mac Finder)

DnD programming: semantics

- Action at drag source: move, copy
  - The default action varies!
  - E.g. Files dragged on the same volume will be moved, files dragged between different volumes wil be copied
  - Taking the alternative action is indicated by e.g. Ctrl (Windows) and Alt (Mac)
    - Transparent at the Java API level
- At drop target:
  - Insert between two elements,
  - Or replace the element under the cursor, etc.

DnD between applications

- Most general
- The drag source and drop target widgets have no knowledge of each other
  - Could be programmed with differnet toolkits
- The communication is restricted to
  - Type (flavor) of the transferred object (MIME)
  - Data transferred
- To ensure this minimal interface, many details known at the "drag point" are lost
- Platform support, Java must use native code

Java DnD 1: Home-made

- Simply using listener and paint callbacks
- No need for special APIs
- MouseListener, MouseMotionListener
  - Detect drag gesture
  - Determine drag cursor location
  - Detect drop
- paint() overriding to draw the drag cursor
  - Most efficient if done on a Container (JFrame, JPanel) so it draws over all standard or custom components inside
  - Don't forget to call super.paint() before, so the contained components are actually painted and updated
- Drop response drawn by repaint (or more specialized update, like scroll) of drop target component
  - Restore its appearance when the drag cursor leaves
  - Or when the drag is cancelled

Java DnD 1: Home-made

- Advantages
  - Easy to make and control to the finest detail
  - No new APIs to learn
- Disadvantages
  - Typically the container and the “drag source” and “drop target” components need to know about each other
    - Interdependence, lack of code flexibility
  - No DnD between windows, or applications
  - No integration with Copy-Paste
- … but enough for the lab

Java Dnd 2: AWT 1.1

- When java started supporting “real” inter-application DnD, a very flexible API was introduced
- http://java.sun.com/javase/6/docs/technotes/guides/dragndrop/spec/dnd1.html
- http://www.javaworld.com/javaworld/jw-03-1999/jw-03-dragndrop.html
- Before drag: DragGestureRecognizer
- At the drag source
  - DragSource, DragSourceContext DragSource(Drag/Drop)Event
- At the potential drop target:
  - DropTarget, DropTargetContext, DropTarget(Drag/Drop)Event
  - autoscroll
- Data transfer:
  - Transferable, DataFlavor
  - MIME types of various objects on various platforms

Java DnD 3: AWT 1.1

- Advantages
  - Very flexible
  - Between windows, between applications
  - Integration with Copy/Paste
    - DataFlavor, Transferable
- Disadvantages:
  - Complex API
  - A number of classes to be written for adding DnD support on a component

Java DnD 3: Swing data transfer

- Provides default data transfer behavior for standard Swing components. Configurable:
  - drag action, drop mode, location-sensitive drop
- http://java.sun.com/javase/6/docs/technotes/guides/swing/1.4/dnd.html
- http://java.sun.com/docs/books/tutorial/uiswing/dnd/index.html
- Drag, Drop, Cut/Copy and Paste behavior of a component can **all** be changed mostly with one class, javax.swing.TransferHandler
- Uses parts of AWT 1.1 DnD, but replaces others
- Calls into the UI platform code, to account for differences in drag gesture recognition and other look-and-feel

Java DnD 3: Swing data transfer

- Advantages:
  - For standard components, DnD can be done flexibly
  - For the default behavior, just call setDragEnabled()
- Disadvantages: not yet ready to use in custom components
  - The TransferHandler mechanism tells when a drag occurs over a target but not when it left the target, or when the drag was cancelled
    - getDropLocation() is package-private on JComponent
    - so only Swing classes can override it
    - http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6448332
    - You can use AWT 1.1 DnD: getDropTarget().addDropTargetListener() in addition to defining a Swing TransferHandler
  - TransferHandler has protected methods so it cannot be decorated (wrapped) to add or alter behavior
    - http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4830695
    - Standard component TransferHandlers are part of the platform dependent code so subclassing to alter behavior would need to be done on one class per platform