

File system GUI representation

Hierarchical containers

each file appears in one folder

Single categorization

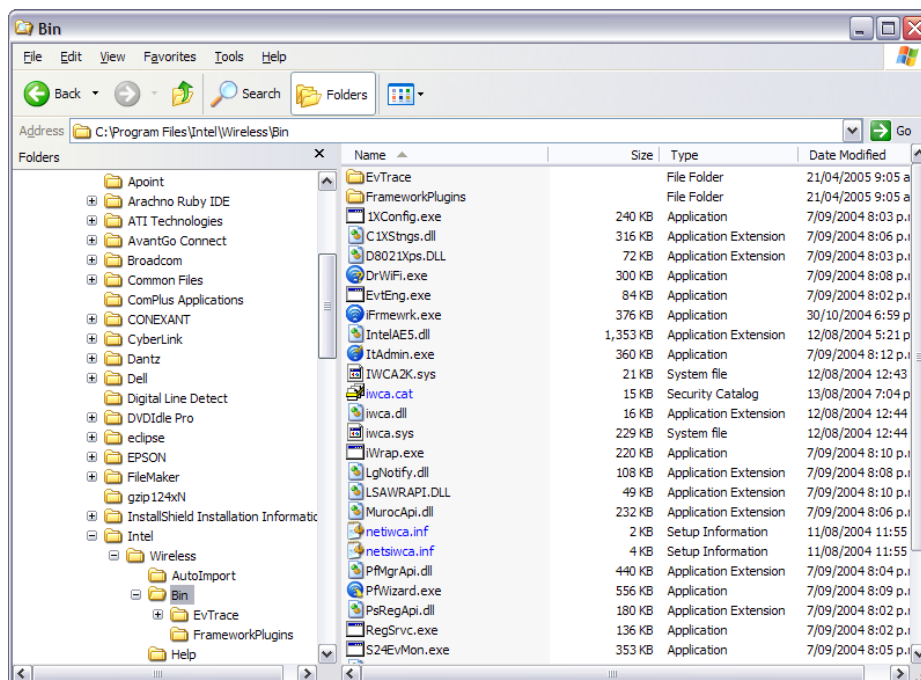
Studies show that people don't use hierarchical file structures properly.

Many files end up in a few common places:

“My Documents” (“Documents” on the Mac)

Desktop

root directory



Multiple categorization

Many documents can easily be stored under different directories.

e.g.,

a document on traffic problems in Auckland
reported in the New Zealand Herald

could be stored under
traffic

or

Auckland

or

Herald

If we are to provide multiple categorization in our OS UIs we have two problems

- Creating and assigning categories
- Retrieving or searching for the information

Assigning categories

A category can be thought of as an attribute of a file. So normal file attributes can be used as categories as well.

We can require/encourage people to add attributes to files.

It is unknown how well people will use the ability to categorize files. (Some initial research results were encouraging, but the test subjects were CS grad students.)

We want the system to be usable by ordinary users.

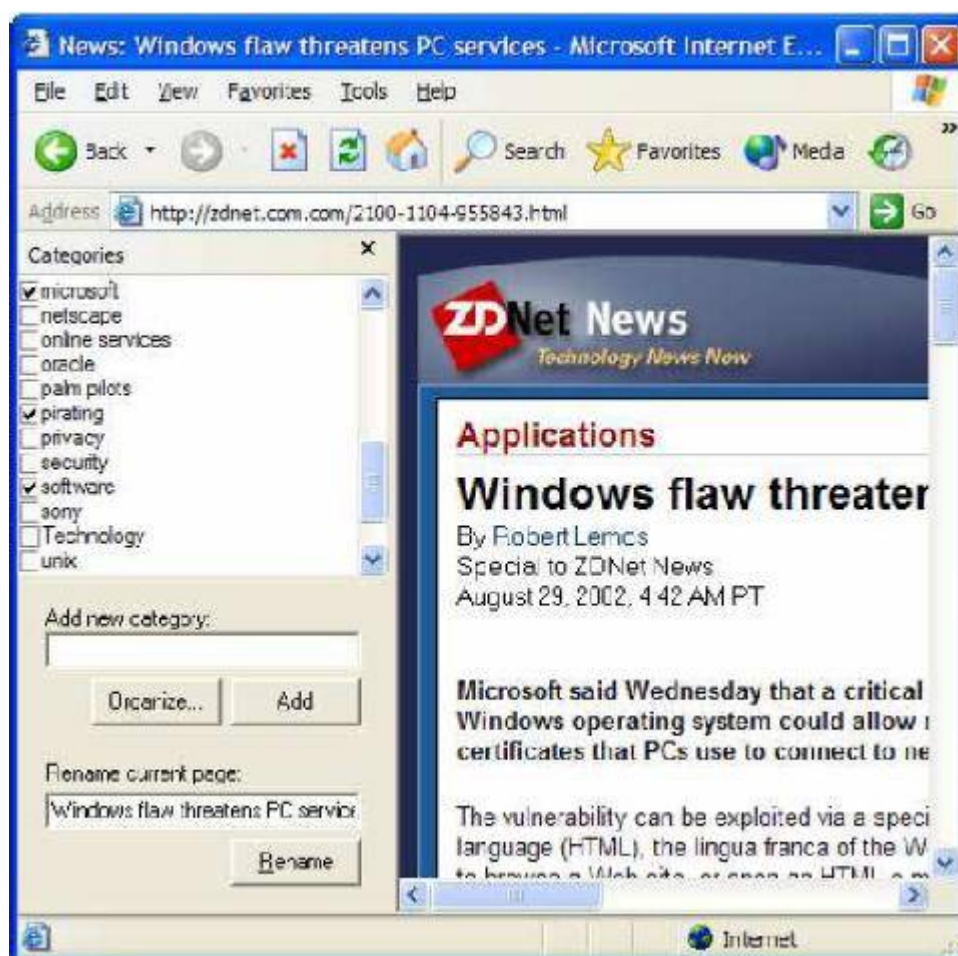
Another solution is to assign categories automatically.

Simple selection

The same UI can be used for web pages, files, appointments, contact lists etc.

In this screenshot you can see several categories. The user ticks the categories the document matches.

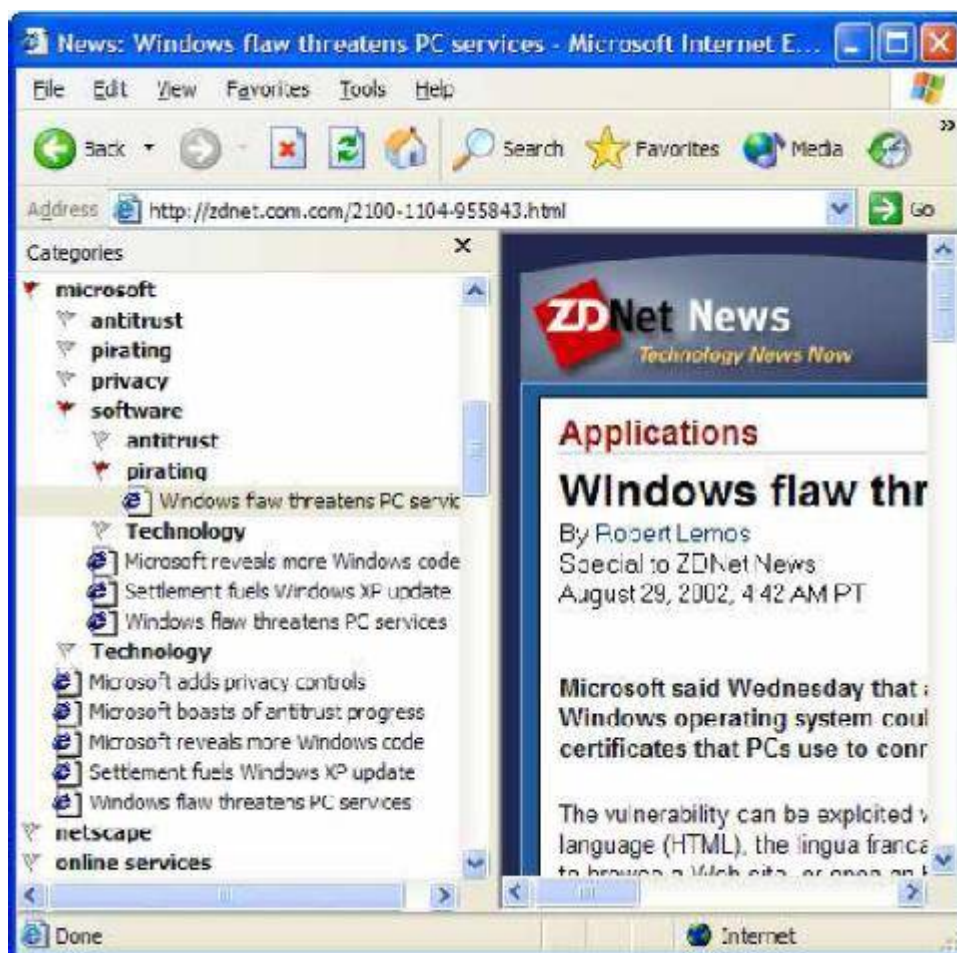
Extra categories can be added easily.



Retrieving

It doesn't matter what category you start looking under. Every category appears as a subcategory of all categories it is paired with.

The example shows web pages but it works exactly the same way for files.



Automatic indexing

There are similar things being done with searching. (Based on web style search engines.)

Rather than have the users worry about where their files are, they just have to be able to describe their contents or some of their attributes.

Indexing engines run regularly, collecting information on the files in the system.

The file system notifies the indexing system whenever a file is added, moves, is altered, or deleted.

A user can then produce a search which appears just like a folder. These searches can be saved. The results are updated dynamically.

“Spotlight” in MacOS 10.4

“WinFS*” eventually in Windows Vista

* Windows Future Storage (I don't know if it will be called that by release time)

Examples

Create a “Smart Folder” or “Library” of all lectures which mention the word “kernel” from all of the courses I have taught in the last 10 years.

A folder with all mail messages sent by a particular person.

All songs written by Peter Gabriel with a length under 3 minutes, that don't have rights management protection.

To make this work

Application programmers will have to provide filters which enable the indexing programs to retrieve relevant information from files. Or include extra attribute information when files are created or altered. Or use open standards for storing file information.

Drag and drop

Another GUI addition. The ability to manipulate objects using their on-screen representation.

Requirements

- drag and drop between different parts of the same application as well as between separate applications
- visual changes of source/destination and cursors to indicate what is happening or allowed
- ways to package information and transform it to be acceptable to other applications

Example drag and drop system

Motif is a UI system built on top of X windows.

Widgets

- text fields
- buttons
- labels
- etc

Widgets can be registered as drag initiators and drop sites

They provide callback procedures (or methods) which are called when drag and drop events are noticed.

There are standard icon changes for the pointer associated with

- the state (is a drop allowed here)
- the operation (move, copy, link)
- the source of the data

Motif dnd (cont.)

Drag sources (initiators)

- package up the data to be transferred in some format
- can monitor the drag operation

Drop targets

- indicate whether they can accept drops of particular data formats (e.g. text, bitmaps)
- indicate whether they accept drops of particular types (copy, move, link)
- provide a way of converting the data to a format it can use

Drag context

- all information about the drag
- source, data, type
- icons associated with the drag – including acceptance and rejection

Drag over effects

- changing the pointer icon (or cursor) to indicate changes in state

Drag under effects

- changing the visual representation of a drop target when the pointer icon is over it

Example action

We will take a hypothetical look at how a simple command “move a file” could be implemented on both CLI and GUI type user interfaces.

Moving a file with a CLI

```
mv fileA directoryB
```

- keystrokes are buffered (and echoed) until the user types enter
- the line gets sent to the shell program
- the shell breaks the line into words
- the first word is checked to see if it is an internal command (it isn't)
- the `mv` program is executed (the remaining two words are sent as parameters) with the current environment including the current working directory
- the `mv` program checks to see if `fileA` exists in the current working directory
- it checks to see if `directoryB` is a directory
- as it is, it moves `fileA` to `directoryB`
- the `mv` program finishes and control returns to the shell

Moving a file with a GUI

Drag `fileA` from one window to the `directoryB` window .

This is the simplified version, lots of checks and state changes aren't represented.

- mouse down is sent
- the lower level of the UI system checks to see where the pointer is - it determines it is over a window
- it passes the mouse down event to the process associated with the folder window including information about which window and where in the window the mouse pointer is
- the process finds what is under that position in that window
- and selects the `fileA` icon – changing its visual representation
- a drag event is detected, the window is a drag source – a drag context is created with appropriate icons and data representing `fileA`
- the pointer cursor changes its icon to represent the drag
- multiple events are generated as the pointer is dragged over the screen to the window representing the destination folder

Moving a file with a GUI (cont.)

- if it goes over a window or widget which is a registered drop target the drop target is asked if it can receive the data (`fileA`)
- drop targets get drag under events, the pointer cursor icon gets changed appropriately
- on releasing the mouse button the destination window gets the drop event
- it determines it can accept `fileA`
- it gets the reference to `fileA` from the drag context and performs the copy
- the starting window process is notified (it might delete the original)
- a `fileA` icon is created in the destination window and removed from the source window
- the drag context is released and the cursor returns to its normal icon

Command files

We frequently want to issue commands from other places or times.

So commands are going to be stored in files and the data in the file is to represent a sequence of commands.

- use the same language to control the system from files as we do from the keyboard
- the user doesn't have to learn two different languages
- problems with trying to write command files for GUIs

For a CLI our command files are going to look like lists of sequential instructions.

The command file language needs to have conditionals

we aren't monitoring the output to make decisions

loops

we commonly want to repeat sequences of actions

variables

to maintain state

Conditionals and Loops

It is simple to make command file languages open command files and start executing the commands in sequential order.

- redirect the normal command interpreter to look at the file rather than at the terminal
- this means the command file language is usually interpreted

It is not much harder to include rudimentary conditional execution.

- a comparison is made
- the next command is executed
- or the command interpreter skips some of the command file until a specific keyword is hit

```
if [ `whoami` != 'robert-s' ]; then
echo 'Now deleting all your files'
else
echo 'Hello Robert.'
fi
```

Loops are done in a similar way. The command interpreter stores lines and re-executes them when it reaches the end of the loop command

Variables

String values

With a CLI since most things are represented by text it makes sense to deal with text strings.

- so we want to be able to compare strings
- do simple string manipulation

Numeric values

It would be nice if you could do some simple work with numbers as well.

- count
- simple arithmetic

Some scripting languages deal with complex object types as well.

Command files for GUIs

AppleScript as an example

Apple designers added a way of communicating with programs known as Apple Events.

- some programs can catch and respond to Apple Events and send Apple Events to other programs
- each program which wants to be scriptable needs to publish commands and object types
- the script is interpreted by the operating system
- it understands some commands and knows what commands the applications understand
- it passes commands on to applications via Apple Events
- the whole scriptable world (those applications which are scriptable) is an object oriented environment
- the objects themselves know what to do with the messages
- there is a huge variety of object classes, ranging from documents to windows to drawings to application programs

Programs that aren't scriptable can still be controlled with low level commands.

Example script

```
display dialog "Import an image file or a folder of images:" buttons {"Cancel", "Folder", "Image"} default button 3
if the button returned of the result is "Folder" then
    set this_item to choose folder
else
    set this_item to choose file of type {"JPEG", "TIFF"}
end if
-- convert the file reference to UNIX style
set this_path to the POSIX path of this_item
tell application "iPhoto" to activate
    tell application "System Events"
        tell process "iPhoto"
            -- open import dialog
            keystroke "I" using {command down, shift down}
            -- summon path input sheet
            keystroke "/" using control down
            delay 1
            tell window "Import Photos"
                tell sheet 1
                    set value of text field 1 to this_path
                    click button 1
                end tell
            click button 1
        end tell
    end tell
end tell
```

This demonstrates both high and low levels of command.