xmonad

A Haskell Success Story

Brent Yorgey

March 22, 2008
FringeDC

Outline

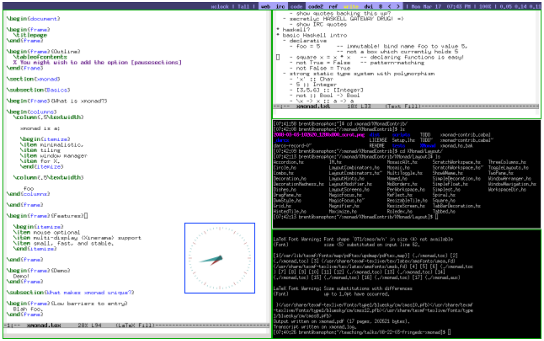
Outline

What is xmonad?

xmonad is a:

- minimalistic,
- tiling
- window manager
- for X.

Original authors: Spencer Janssen, Don Stewart, and Jason Creighton.


wclock | Tall | web irc code code2 ref write dvi 8 < > | Mon Mar 17 07:45 PM | 100% | 0.05 0.14 0.11

```
\begin{document}

\begin{frame}
  \titlepage
\end{frame}

\begin{frame}{Outline}
  \tableofcontents
  % You might wish to add the option [pausesections]
\end{frame}

\section{xmonad}

\subsection{Basics}

\begin{frame}{What is xmonad?}

\begin{columns}
  \column{.5\textwidth}

    xmonad is a:

    \begin{itemize}
    \item minimalistic,
    \item tiling
    \item window manager
    \item for X.
    \end{itemize}

  \column{.5\textwidth}

     Foo
\end{columns}

\end{frame}

\begin{frame}{Features}

  \begin{itemize}
  \item mouse optional
  \item multi-display (Xinerama) support
  \item small, fast, and stable.
  \end{itemize}

\end{frame}

\begin{frame}{Demo}
  Demo!
\end{frame}

\subsection{What makes xmonad unique?}

\begin{frame}{Low barriers to entry}
  Blah foo.
\end{frame}
-1:--  xmonad.tex      28% L94    (LaTeX Fill)----------------------------
```

```
      - show quotes backing this up?
    - secretly: HASKELL GATEWAY DRUG! =)
      - show IRC quotes
  * haskell?
  * basic Haskell intro
    - declarative
      - foo = 5       -- immutable! bind name foo to value 5,
                      -- not a box which currently holds 5
      - square x = x * x    -- declaring functions is easy!
      - not True = False    -- pattern-matching
      - not False = True
    - strong static type system with polymorphism
      - 'x' :: Char
      - 5 :: Integer
      - [3,5,6] :: [Integer]
      - not :: Bool -> Bool
      - \x -> x :: a -> a
--:--  xmonad.txt      18% L33    (Text Fill)----------------------------
```

```
[07:41:58 brent@xenophon:~]$ cd xmonad/XMonadContrib/
[07:42:08 brent@xenophon:~/xmonad/XMonadContrib]$ ls
2008-03-05-102620_1280x800_scrot.png  dist      scripts   TODO    xmonad-contrib.cabal
_darcs                                LICENSE   Setup.lhs TODO~   xmonad-contrib.cabal~
darcs-record-0~                       README    tests     XMonad  xmonad.hs.bak
[07:42:09 brent@xenophon:~/xmonad/XMonadContrib]$ cd XMonad/Layout/
[07:42:13 brent@xenophon:~/xmonad/XMonadContrib/XMonad/Layout]$ ls
Accordion.hs           IM.hs                  MosaicAlt.hs       ScratchWorkspace.hs    ThreeColumns.hs
Circle.hs              LayoutCombinators.hs   Mosaic.hs          ScratchWorkspace.hs~   ToggleLayouts.hs
Combo.hs               LayoutCombinators.hs~  MultiToggle.hs     ShowWName.hs           TwoPane.hs
Decoration.hs          LayoutHints.hs         Named.hs           SimpleDecoration.hs    WindowArranger.hs
DecorationMadness.hs   LayoutModifier.hs      NoBorders.hs       SimpleFloat.hs         WindowNavigation.hs
Dishes.hs              LayoutScreens.hs       PerWorkspace.hs    Simplest.hs            WorkspaceDir.hs
DragPane.hs            MagicFocus.hs          Reflect.hs         Spiral.hs
DwmStyle.hs            MagicFocus.hs~         ResizableTile.hs   Square.hs
Grid.hs                Magnifier.hs           ResizeScreen.hs    TabBarDecoration.hs
HintedTile.hs          Maximize.hs            Roledex.hs         Tabbed.hs
[07:42:13 brent@xenophon:~/xmonad/XMonadContrib/XMonad/Layout]$ 
```

```
LaTeX Font Warning: Font shape `OT1/cmss/m/n' in size <4> not available
(Font)              size <5> substituted on input line 62.

[1{/var/lib/texmf/fonts/map/pdftex/updmap/pdftex.map}] (./xmonad.toc) [2]
(./xmonad.toc) [3] (/usr/share/texmf-texlive/tex/latex/amsfonts/umsa.fd)
(/usr/share/texmf-texlive/tex/latex/amsfonts/umsb.fd) [4] [5] [6] (./xmonad.toc
) [7] [8] [9] [10] [11] [12] (./xmonad.toc) [13] (./xmonad.toc) [14]
(./xmonad.toc) [15] (./xmonad.toc) [16] (./xmonad.toc) [17] (./xmonad.aux)

LaTeX Font Warning: Size substitutions with differences
(Font)              up to 1.0pt have occurred.

 )</usr/share/texmf-texlive/fonts/type1/bluesky/cm/cmss10.pfb></usr/share/texmf
-texlive/fonts/type1/bluesky/cm/cmss12.pfb></usr/share/texmf-texlive/fonts/type
1/bluesky/cm/cmss8.pfb>
Output written on xmonad.pdf (17 pages, 202621 bytes).
Transcript written on xmonad.log.
[07:40:26 brent@xenophon:~/teaching/talks/08-22-03-fringedc-xmonad]$ 
```

Features

- multiple workspaces
- mouse optional
- multi-display (Xinerama) support
- small, fast, and stable
- recommended by four out of five plush penguins

Programming in Haskell

Demo: basic xmonad features

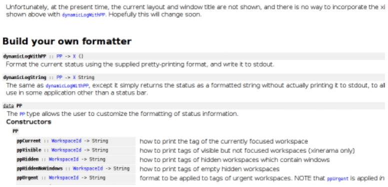
Demo!

Outline

Low barriers to use

Switching to xmonad is easy:

- friendly user community
- tons of documentation



Unfortunately, at the present time, the current layout and window title are not shown, and there is no way to incorporate the xi
shown above with `dynamicLogWithPP`. Hopefully this will change soon.

Build your own formatter

`dynamicLogWithPP :: PP -> X ()`

Format the current status using the supplied pretty-printing format, and write it to stdout.

`dynamicLogString :: PP -> X String`

The same as `dynamicLogWithPP`, except it simply returns the status as a formatted string without actually printing it to stdout, to al
use in some application other than a status bar.

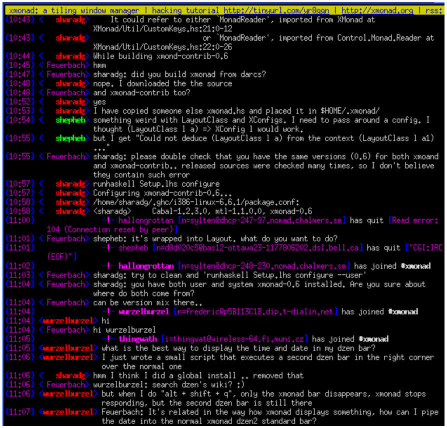
`data PP`

The `PP` type allows the user to customize the formatting of status information.

Constructors

`PP`

`ppCurrent :: WorkspaceId -> String`	how to print the tag of the currently focused workspace
`ppVisible :: WorkspaceId -> String`	how to print tags of visible but not focused workspaces (xinerama only)
`ppHidden :: WorkspaceId -> String`	how to print tags of hidden workspaces which contain windows
`ppHiddenNoWindows :: WorkspaceId -> String`	how to print tags of empty hidden workspaces
`ppUrgent :: WorkspaceId -> String`	format to be applied to tags of urgent workspaces. NOTE that `ppUrgent` is applied in



```
xmonad: a tiling window manager | hacking tutorial http://tinyurl.com/yr8ggn | http://xmonad.org | rss:
(10:43) <   sharadg>     It could refer to either `MonadReader', imported from XMonad at
                   XMonad/Util/CustomKeys.hs:21:0-12
(10:43) <   sharadg>                           or `MonadReader', imported from Control.Monad.Reader at
                   XMonad/Util/CustomKeys.hs:22:0-26
(10:44) <   sharadg> While building xmond-contrib-0.6
(10:45) < Feuerbach> hmm
(10:47) < Feuerbach> sharadg: did you build xmonad from darcs?
(10:48) <   sharadg> nope. I downloaded the the source
(10:48) < Feuerbach> and xmonad-contrib too?
(10:52) <   sharadg> yes
(10:53) <   sharadg> I have copied someone else xmonad.hs and placed it in $HOME/.xmonad/
(10:54) <   shepheb> something weird with LayoutClass and XConfigs. I need to pass around a config. I
                   thought (LayoutClass l a) => XConfig l would work.
(10:55) <   shepheb> but I get "Could not deduce (LayoutClass l a) from the context (LayoutClass l a1)
                   ..."
(10:55) < Feuerbach> sharadg: please double check that you have the same versions (0.6) for both xmoand
                   and xmonad-contrib.. released sources were checked many times, so I don't believe
                   they contain such error
(10:57) <   sharadg> runhaskell Setup.lhs configure
(10:57) <   sharadg> Configuring xmonad-contrib-0.6...
(10:58) <   sharadg> /home/sharadg/.ghc/i386-linux-6.6.1/package.conf:
(10:58) <   sharadg> <sharadg>     Cabal-1.2.3.0, mtl-1.1.0.0, xmonad-0.6
(11:00)              -!- hallongrottan [n=sylten@dhcp-247-97.nomad.chalmers.se] has quit [Read error:
          104 (Connection reset by peer)]
(11:01) < Feuerbach> shepheb: it's wrapped into Layout. what do you want to do?
(11:01)              -!- shepheb [n=d8d020c5@bas12-ottawa23-1177806202.dsl.bell.ca] has quit ["CGI:IRC
          (EOF)"]
(11:02)              -!- hallongrottan [n=sylten@dhcp-248-230.nomad.chalmers.se] has joined #xmonad
(11:03) < Feuerbach> sharadg: try to clean and 'runhaskell Setup.lhs configure --user'
(11:04) < Feuerbach> sharadg: you have both user and system xmonad-0.6 installed. Are you sure about
                   where do both come from?
(11:04) < Feuerbach> can be version mix there..
(11:04)              -!- wurzelburzel [n=frederic@p5B113C1B.dip.t-dialin.net] has joined #xmonad
(11:04) <wurzelburzel> hi
(11:04) < Feuerbach> hi wurzelburzel
(11:05)              -!- thingwath [i=thingwat@wireless-64.fi.muni.cz] has joined #xmonad
(11:05) <wurzelburzel> what is the best way to display the time and date in my dzen bar?
(11:06) <wurzelburzel> I just wrote a small script that executes a second dzen bar in the right corner
                     over the normal one
(11:06) <   sharadg> hmm I think I did a global install .. removed that
(11:06) < Feuerbach> wurzelburzel: search dzen's wiki? :)
(11:06) <wurzelburzel> but when I do "alt + shift + q", only the xmonad bar disappears, xmonad stops
                     responding, but the second dzen bar is still there
(11:07) <wurzelburzel> Feuerbach: It's related in the way how xmonad displays something, how can I pipe
                     the date into the normal xmonad dzen2 standard bar?
```

Secret sauce: Haskell

- Written in Haskell
- Easy to customize

* No artificial flavors, references, continuations or global side-effects.
IO, state and related features provided in controlled monadic contexts.
Users must ensure that monads obey all relevant equational laws.
Programmer's use of unsafePerformIO voids any and all warranties.

Secret sauce: Haskell

- Written in Haskell
- Easy to customize
    - . . . in Haskell

Haskell
100% Pure*
Functional
Programming
* No artificial flavors, references, continuations or global side-effects.
IO, state and related features provided in controlled monadic contexts.
Users must ensure that monads obey all relevant equational laws.
Programmer's use of unsafePerformIO voids any and all warranties.

Secret sauce: Haskell

- Written in Haskell
- Easy to customize
  - ...in Haskell
  - ...by users who don't know Haskell!

Haskell
100% Pure* Functional Programming
* No artificial flavors, references, continuations or global side-effects. IO, state and related features provided in controlled monadic contexts. Users must ensure that monads obey all relevant equational laws. Programmer's use of unsafePerformIO voids any and all warranties.

An unsolicited quote

> *I may not know much (really, any) Haskell itself, but employing it as a "configuration language" is certainly far easier than anyone might give it credit. . . . the modules are perhaps the best bit of documentation I've seen in any code, ever. Additionally, you get a free IRC room full of other xmonad users who are always more than willing to point you in the right direction . . . .*

— Will Farrington (`wfarr`), March 16, 2008

`http://dev.compiz-fusion.org/~wfarr/viewpost?id=5`

xmonad: Haskell gateway drug!

Outline

What is Haskell?

- born in 1987
- named for the logician Haskell Curry
- research language, but also practical
- increasingly popular!
- functional, strongly typed, pure, lazy

Outline

Haskell is declarative

- names bind to values
- no mutation

Example

```
n = 6
```

Haskell is declarative

- names bind to values
- no mutation

Example

```
n = 6
n = 7     -- ERROR
```

Haskell has strong static typing

- every value has a type
- types cannot be mixed
- types checked at compile time

Example

```
'x'        :: Char
5          :: Integer
"xmonad"   :: String      -- [Char]
[3,5,6]    :: [Integer]
not        :: Bool -> Bool
safeHead   :: [a] -> Maybe a
```

Haskell has user-defined algebraic data types

Example

```
data Color = Red | Green | Cerulean
data List a = Nil | Cons a (List a)
data Tree a = Empty | Node a (Tree a) (Tree a)
```

Example

```
data Workspace i l a =
    Workspace { tag    :: !i
              , layout ::  l
              , stack  :: Maybe (Stack a) }
data Stack a = Stack { focus  :: !a
                     , up     :: [a]
                     , down   :: [a] }
```

Haskell is functional

- data-oriented, not control-oriented
- first-class functions

Example

```
filter :: (a -> Bool) -> [a] -> [a]
filter p []     = []
filter p (x:xs) = if (p x)
                    then x : filter p xs
                    else     filter p xs
```

Haskell is lazy

- expressions not evaluated until needed
- enables optimization, computing with infinite data structures

Example

```
> take 10 [1..]
[1,2,3,4,5,6,7,8,9,10]

> let fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
> take 15 fibs
[0,1,1,2,3,5,8,13,21,34,55,89,144,233,377]
```

Haskell is pure

Core idea: types tell you all you need to know

- output depends only on input
- no “side effects”

Haskell is pure

Core idea: types tell you all you need to know

- output depends only on input
- no “side effects”

Example

```
f :: Int -> Int
```

Haskell is pure

Core idea: types tell you all you need to know

- output depends only on input
- no "side effects"

Example

```
f :: Int -> Int

-- BAD:
f n = getTemperature + n
```

Haskell is pure

Core idea: types tell you all you need to know

- output depends only on input
- no "side effects"

Example

```
f :: Int -> Int

-- ALSO BAD:
f n = releaseEvilMonkeys; return (n+2)
```

Haskell is pure

Core idea: types tell you all you need to know

- output depends only on input
- no “side effects”

Example

```
f :: Int -> Int

-- much better!
f n = (n^2 + 1) * 2
```

Advantages of purity

- easier to reason about and safely transform programs

Example

```
f x = foo x + bar (foo x)

g x = y + bar y
    where y = foo x
```

- easier to test
- makes laziness possible

But we want effects!

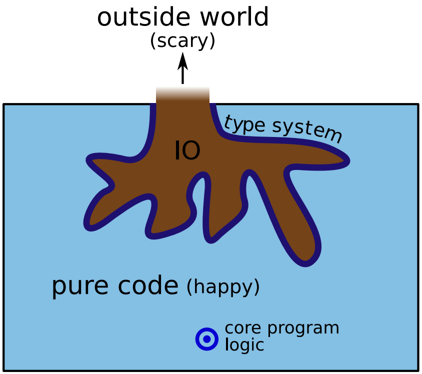
- ultimate purpose of programs is to have effects!
- programs with no effects just “make the box get hot”
- need a way to separate effectful from pure code. . .

Solution: monads!

- concept from category theory
- warm and fuzzy
- basic idea: computations which can be composed to form larger computations
- one application: represent computations which may have side effects when run (IO monad)

Solution: monads!

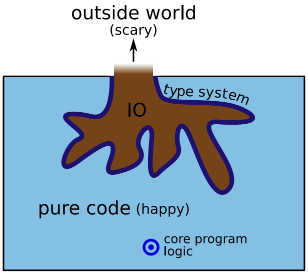
- type system separates values from computations which produce values (and may have side effects!)



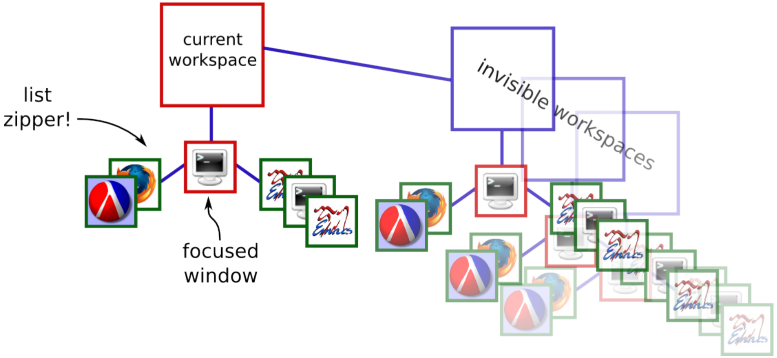
Outline

The xmonad core

- appropriate, specific data structures to hold state
- pure operations to manipulate data structures
- invariant properties of operations specified with QuickCheck
- custom X monad (IO + WM state + configuration) to encapsulate side-effecting operations



The xmonad core



Testing with QuickCheck

- properties verified with random data
- HPC ensures code coverage
- xmonad core correct with high probability!

Example

```
-- shifting focus is trivially reversible
prop_focus_left  (x :: T) = (focusUp   (focusDown x)) == x
prop_focus_right (x :: T) = (focusDown (focusUp   x)) == x

-- focus master is idempotent
prop_focusMaster_idem (x :: T) =
    focusMaster x == focusMaster (focusMaster x)
```

Purity FTW!

- way more purity than you think
- separating pure/impure $\Longrightarrow$ big win in specification and testing
- future xmonad work: test impure parts by swapping in pure IO replacement!

Outline

Configuring xmonad

- configuration goes in `$HOME/.xmonad/xmonad.hs`
- `xmonad.hs` is a Haskell program!
- "write your own window manager" using the provided xmonad libraries
- don't let the power go to your head

Demo: configuring xmonad

Demo!

Outline

Haskell: win for modularity

- Haskell functional & pure $\Longrightarrow$ easy to make xmonad very modular
- xmonad core very small (just over 1K LOC)
- most interesting functionality is in the extension library!

xmonad-contrib extension library

- huge library of extensions (118 modules, over 6500 LOC)
- contributions from ∼60 different authors

Accordion Anneal AppendFile Arossato Circle Combo
Commands Configuring ConstrainedResize CopyWindow CustomKeys CycleSelectedLayouts
CycleWS Decoration DecorationMadness DeManage Developing Directory
DirExec Dishes Dmenu Doc Dons DragPane
Droundy DwmPromote DwmStyle DynamicLog DynamicWorkspaces Dzen
Email EwmhDesktops Extending EZConfig FindEmptyWorkspace FlexibleManipulate
FlexibleResize FloatKeys FocusNth Fontc Grid HintedTile
IM Input Invisible Layout LayoutCombinators LayoutHints
LayoutModifier LayoutScreens Loggers MagicFocus Magnifier Man
ManageDocks ManageHelpers Maximize Mosaic MosaicAlt MouseGestures
MouseResize MultiToggle Named NamedWindows NoBorders NoBorders
PerWorkspace PerWorkspaceKeys Promote Prompt Reflect ResizableTile
ResizeScreen Roledex RotSlaves RotView Run Scratchpad
ScratchWorkspace Search SetWMName Shell ShowWName SimpleDate
SimpleDecoration SimpleFloat Simplest SinkAll Sjanssen Spiral
Square Ssh Submap SwapWorkspaces TabBarDecoration Tabbed
TagWindows Theme Themes ThreeColumns Timer ToggleLayouts
TwoPane UpdatePointer UrgencyHook Warp Window WindowArranger
WindowBringer WindowGo WindowNavigation WindowProperties Workspace WorkspaceCompare
WorkspaceDir XMonad XPropManage XSelection XUtils

Demo: extending xmonad

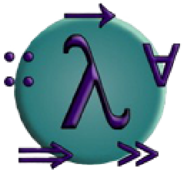
Demo!

Summary

- xmonad's success due in large part to Haskell
- purity makes testing easy!
- Haskell as configuration language makes configuring/extending xmonad fun
- try it, you'll like it. . .

Want to learn more Haskell?

- #haskell IRC channel on freenode.net (`http://haskell.org/haskellwiki/IRC_channel`)
- Some tutorials:
  - `http://en.wikibooks.org/wiki/Haskell`
  - `http://darcs.haskell.org/yaht/yaht.pdf`
- Haskell wiki: `http://www.haskell.org/`



Want to try xmonad?

- xmonad web page, with downloads, tutorials, and documentation: `http://xmonad.org`
- #xmonad IRC channel on freenode.net