Userland

creating an
integrated dataflow environment
for end-users

Hisham Muhammad
h@hisham.hm
@hisham_hm

LIVE 2019 - Athens, Greece

A quick introduction

PhD at PUC-Rio

programming languages group: LabLua

sneaking into HCI classes (EUD, systems thinking...)

thesis: “Dataflow Semantics for End-User Programmable Applications” - case studies on LabVIEW, Pure Data, Excel

An itch that goes way back

General insatisfaction with computing

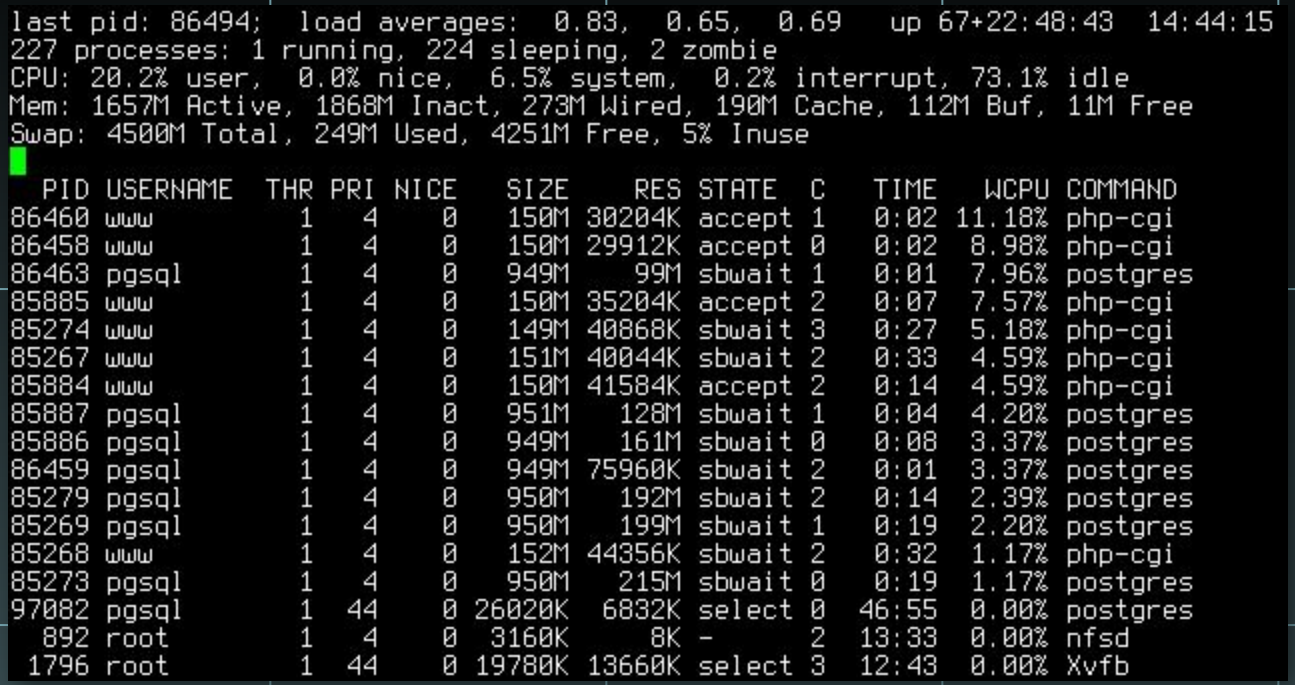
that love/hate feel that drives people into PL research

Deep nostalgia for the Apple II days

that sense of ownership and power that users have lost

A deeply ingrained OSS ethos

if you don’t like it, change it


last pid: 86494; load averages: 0.83, 0.65, 0.69   up 67+22:48:43 14:44:15
227 processes: 1 running, 224 sleeping, 2 zombie
CPU: 20.2% user, 0.0% nice, 6.5% system, 0.2% interrupt, 73.1% idle
Mem: 1657M Active, 1868M Inact, 273M Wired, 190M Cache, 112M Buf, 11M Free
Swap: 4500M Total, 249M Used, 4251M Free, 5% Inuse
PID USERNAME THR PRI NICE SIZE RES STATE C TIME WCPU COMMAND
86460 www 1 4 0 150M 30204K accept 1 0:02 11.18% php-cgi
86458 www 1 4 0 150M 29912K accept 0 0:02 8.98% php-cgi
86463 pgsql 1 4 0 949M 99M sbwait 1 0:01 7.96% postgres
85885 www 1 4 0 150M 35204K accept 2 0:07 7.57% php-cgi
85274 www 1 4 0 149M 40868K sbwait 3 0:27 5.18% php-cgi
85267 www 1 4 0 151M 40044K sbwait 2 0:33 4.59% php-cgi
85884 www 1 4 0 150M 41584K accept 2 0:14 4.59% php-cgi
85887 pgsql 1 4 0 951M 128M sbwait 1 0:04 4.20% postgres
85886 pgsql 1 4 0 949M 161M sbwait 0 0:08 3.37% postgres
86459 pgsql 1 4 0 949M 75960K sbwait 2 0:01 3.37% postgres
85279 pgsql 1 4 0 950M 192M sbwait 2 0:14 2.39% postgres
85269 pgsql 1 4 0 950M 199M sbwait 1 0:19 2.20% postgres
85268 www 1 4 0 152M 44356K sbwait 2 0:32 1.17% php-cgi
85273 pgsql 1 4 0 950M 215M sbwait 0 0:19 1.17% postgres
97082 pgsql 1 44 0 26020K 6832K select 0 46:55 0.00% postgres
892 root 1 4 0 3160K 8K - 2 13:33 0.00% nfsd
1796 root 1 44 0 19780K 13660K select 3 12:43 0.00% Xvfb

```
  0  [|||||                                    8.4%]   Tasks: 94, 42 thr; 1 running
  1  [|||                                      3.8%]   Load average: 0.47 0.26 0.22
  2  [|                                        0.6%]   Uptime: 04:28:22
  3  [||                                       1.3%]   Battery: n/a
  Mem[||||||||||||||||||||||||||||||||||1114/3819MB]
  Swp[                                       0/0MB]

IO Priority:              PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM% Command
None (based on nice)        1 root       40   0  1672   572   508 S  0.0  0.0 init [2]
Realtime 0 (High)       15814 hisham     40   0 16140  6284  3856 S  0.0  0.2 ├─ urxvt -cr green -fn *-lode-* -fb *
Realtime 1              15815 hisham     40   0  8804  6004  1692 S  0.0  0.2 │  └─ zsh
Realtime 2              15836 hisham     40   0 45512 30796 13920 S  0.0  0.8 │     └─ gimp
Realtime 3              16474 hisham     40   0 26020 12624  8792 S  0.6  0.3 │        ├─ /System/Index/lib/gimp/2.
Realtime 4              16047 hisham     40   0 20552  5344  3236 S  0.0  0.1 │        └─ /System/Index/lib/gimp/2.
Realtime 5              15345 hisham     40   0 62852 54784  3920 S  0.0  1.4 ├─ urxvt -cr green -fn *-lode-* -fb *
Realtime 6              15346 hisham     40   0  9132  6460  1868 S  0.0  0.2 │  └─ zsh
Realtime 7 (Low)        15357 hisham     40   0  1716   564   468 T  0.0  0.0 │     └─ cw: wrapping [find] {pid=153
Best-effort 0 (High)    15358 hisham     40   0 11768  9128   756 S  0.0  0.2 │        └─ find
Best-effort 1           15291 hisham     40   0 16332  6492  3920 S  0.0  0.2 ├─ urxvt -cr green -fn *-lode-* -fb *
Best-effort 2           15292 hisham     40   0  9000  6316  1856 S  0.0  0.2 │  └─ zsh
Best-effort 3           15340 hisham     40   0  3116  1852  1148 R  4.4  0.0 │     └─ ./htop
Best-effort 4           14628 hisham     40   0 16140  6304  3864 S  0.0  0.2 ├─ urxvt -cr green -fn *-lode-* -fb *
Best-effort 5           14629 hisham     40   0  9240  6496  1868 S  0.0  0.2 │  └─ zsh
Best-effort 6           14646 hisham     40   0  1712   548   468 S  0.0  0.0 │     └─ cw: wrapping [env] {pid=1464
Best-effort 7 (Low)     14647 hisham     40   0  5644  2620  1204 S  0.0  0.1 │        └─ /bin/bash /System/Links/E
Idle                    16439 hisham     40   0  8012  5340  2348 D  0.6  0.1 │           └─ python /System/Links/E
                        13475 hisham     40   0 16140  6380  3864 S  0.0  0.2 ├─ urxvt -cr green -fn *-lode-* -fb *
                        13476 hisham     40   0  8988  6320  1868 S  0.0  0.2 │  └─ zsh
                        13808 hisham     40   0  3976  1912  1532 S  0.0  0.0 │     └─ ssh -t loderunner,htop@shell
                        13384 hisham     40   0 16140  6288  3864 S  0.0  0.2 ├─ urxvt -cr green -fn *-lode-* -fb *
                        13385 hisham     40   0  9000  6296  1844 S  0.0  0.2 │  └─ zsh
                        10544 hisham     40   0 26212 16596  3920 S  0.0  0.4 ├─ urxvt -cr green -fn *-lode-* -fb *
EnterSet   EscCancel
```

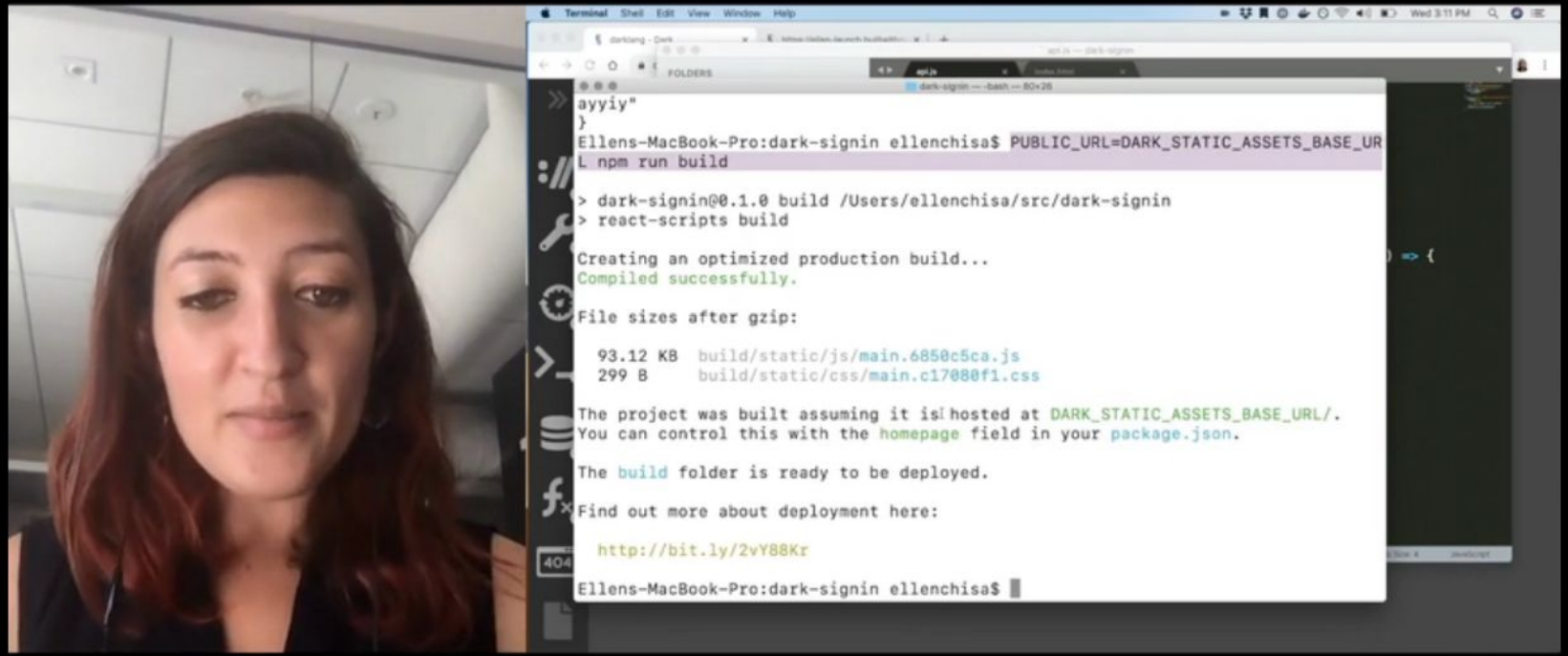
That sense of ownership and power

Divide between end-users and programmers

we still have that power! but we took it away from users

But we are still on super clunky tools

a screenful of terminal emulators? really?



And, now I'm going to use
Dark's command line application

Can we do better?

Lessons learned from success stories

Programmable environments users love

LabVIEW, Pure Data (Max/MSP), Excel

What do they have in common?

A SMALL MATTER OF PROGRAMMING

PERSPECTIVES ON END USER COMPUTING

BONNIE A. NARDI

End user	formulas	macro recorder	node editor	Unix shell	level editor
Domain dev	macros	textual macros	scripting	shell script	game scripting
Core app	spreadsheet	word processor	3D app	C utilities	video game

Figure 2.1: Nardi's three types of programmers and three-layer architectures in end-user programmable systems

an produce a text document with a very complex layout in LibreOffice Writer or intricate multi-layer in Gimp without their scripti

Design dimension	Design alternatives
Box-line representation	no; yes
Iteration	no; limited; yes (cycles); yes (construct)
Subprogram abstraction	no; yes
Selector/distributor	no; yes
Flow of data	uni-directional; bi-directional
Sequence construct	no; yes
Type checking	no; yes (limited); yes (all types)
Higher-order functions	no; yes
Execution mode	data-driven; demand-driven
Liveness level	1 (informative); 2 (significant); 3 (responsive); 4 (live)

Table 3.1: Hils's design dimensions for dataflow visual languages

Design dimension	Design alternatives
Dataflow model	static; dynamic
N-to-1 inputs	no; yes (auto-merge); yes (queueing)
Time-dependent firing	no; yes
Rate-based evaluation	no; synchronous; cyclo-static; quasi-static; dynamic
Separate program and UI	no; yes
Indirect connections	no; yes (static); yes (runtime-evaluated)
Textual sub-language	no; yes (functional); yes (imperative)

Table 3.2: Additional design dimensions for dataflow end-user languages

whether its dataflow model is static or dynamic, according to definitions presented in Secti

General information	**Pure Data**	**Excel**	**LabVIEW**	**Reaktor**	**VEE**	**Blender**
Main reference	[P+15]		[Nat01]	[Nat15]	[Agi11]	[Blc17]
Licensing	3-clause BSD	Proprietary	Proprietary	Proprietary	Proprietary	GNU GPL v2+
Initial release	1996	1985	1986	1999	1991	1995
Latest release	2016	2016	2016	2015	2013	2017
Application domain	Music	Office	Engineering	Music	Engineering	3D graphics

Design alternatives [Hil92]	**Pure Data**	**Excel**	**LabVIEW**	**Reaktor**	**VEE**	**Blender**
Box-line representation	Yes	No	Yes	Yes	Yes	Yes
Iteration	Yes (cycles)	Limited	Yes (construct)	Limited	Yes	No
Subprogram abstraction	Yes	No	Yes	Yes	Yes	Yes
Selector/distributor	Yes	Yes	Yes	Yes	Yes	Yes
Flow of data	Uni	Uni	Uni	Uni	Uni	Uni
Sequence construct	No	No	Yes	No	Yes	No
Type checking	Limited	No	Yes	Yes	No	Yes
Higher-order functions	No	No	No	No	No	No
Execution mode	Data-driven	Demand-driven	Data-driven	Demand-driven	Data-driven	Data-driven
Liveness level [Tan90]	2	3	2	2	2	3

Additional design alternatives	**Pure Data**	**Excel**	**LabVIEW**	**Reaktor**	**VEE**	**Blender**
Dataflow model	Dynamic	Static	Static	Static	Static	Static
N-to-1 inputs	Yes	No	No	No	No	No
Separate edit/use views	No	No	Yes	Yes	Yes	No
Time-dependent firing	Yes	No	Yes	Yes	Yes	No
Rate-based evaluation	Synchronous	No	No	Synchronous	No	No
Indirect connections	Yes	Yes	Yes	Yes	Yes	No
Dynamic connections	Yes	Yes	Yes	No	No	No
Textual sub-language	Imperative	Functional	Imperative	No	Imperative	No
Scripting	Python, Lua	VBA	MATLAB	Reaktor Core	MATLAB	OSL, Python

Table 8.1: A comparison of contemporary dataflow UI-level languages

Userland

Integrated Dataflow Environment

inspired by the common core of dataflow apps

a “shell” for multiple types of applications

implemented as modules (“mods”)

Demo time!

Current status

Ongoing prototype written in Lua

using Love2D graphics engine

Application modules in Lua

spreadsheet, shell, synthesizer

Fleshing out details of the semantics

Future directions

Robust core

“rewrite it in Rust?”

Stable API for application modules

UI improvements for multiple modes of interaction

mouse/touch friendliness, etc.

Thank you!

https://userland.org

hisham@userland.org
@hisham_hm

Appendix: Classification, as of today

Box-line representation: not yet (intended)
Iteration: not yet (intended)
Subprogram abstraction: not yet (intended)
Selector/distributor: sort of (shell)
Flow of data: Unidirectional
Sequence construct: no (not yet?)
Type checking: no (not yet?)
Higher order functions: no
Execution mode: data and demand-driven
Liveness level: 4 (live)

Dataflow model: static
N-to-1 inputs: no
Separate edit/use views: no
Time-dependent firing: yes (synth)
Rate-based evaluation: synchronous (synth, shell)
Indirect connections: no
Dynamic connections: no (not yet?)
Textual sub-language: imperative (shell), functional (spreadsheet)
Scripting: Lua