Dataflow Semantics for End-User Programmable Applications

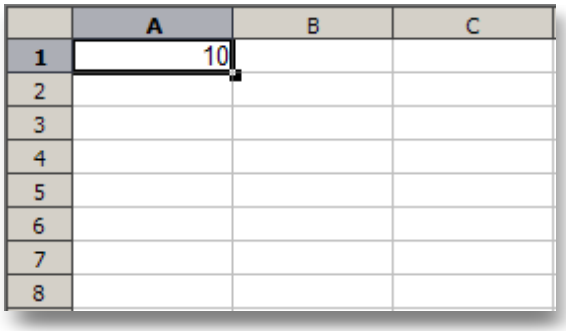
Hisham Muhammad

April 28, 2017

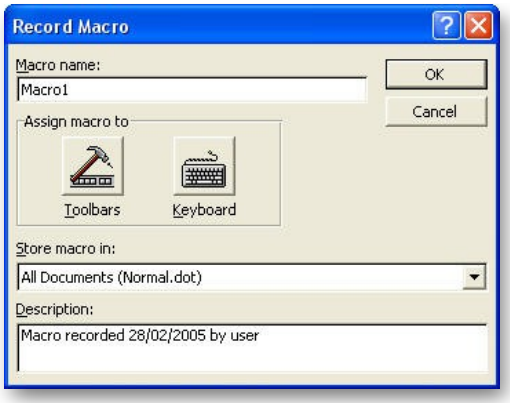
End-user programming

End-user programming

spreadsheets



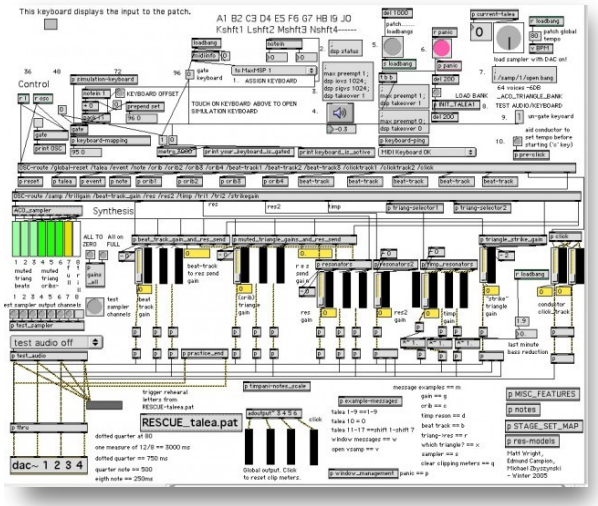
macros



scripting

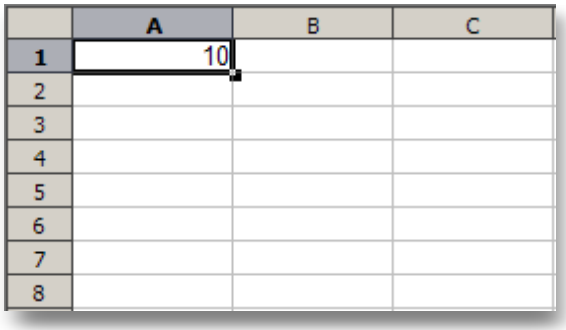
```
AddEntityCollideCallback("Player","AreaHelp",
   "CollidePlayerWithAreaHelp");

void CollidePlayerWithAreaHelp()
{
    if(HasItem("cellar_key") == true)
    {
        SetMessage("Use your inventory to select "+
        "key and use on door.", 10);
    }
    else
    {
        SetMessage("You will need a key to open "+
        "the door.", 5);
    }

}
```

node editors

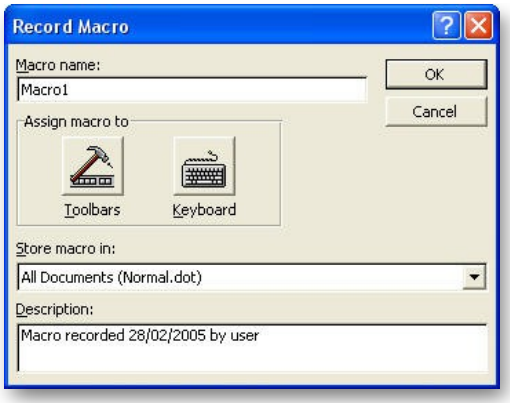


End-user programming

spreadsheets



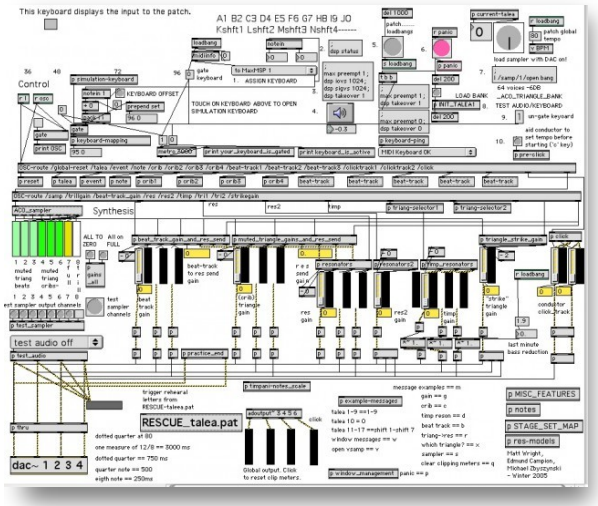
macros



scripting

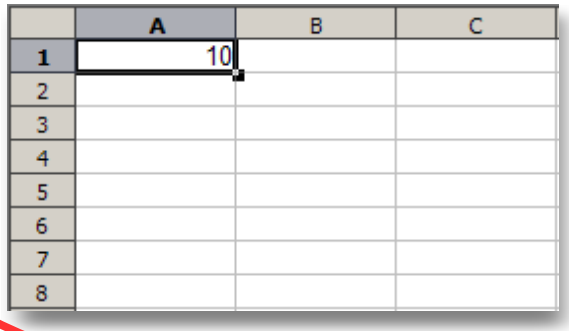
```
AddEntityCollideCallback("Player","AreaHelp",
   "CollidePlayerWithAreaHelp");

void CollidePlayerWithAreaHelp()
{
    if(HasItem("cellar_key") == true)
    {
        SetMessage("Use your inventory to select "+
        "key and use on door.", 10);
    }
    else
    {
        SetMessage("You will need a key to open "+
        "the door.", 5);
    }
}
```

node editors

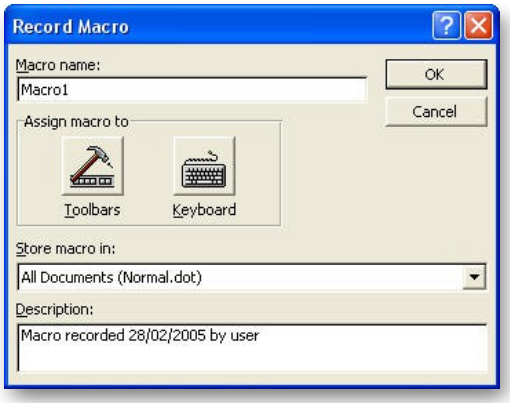


End-user programming

spreadsheets



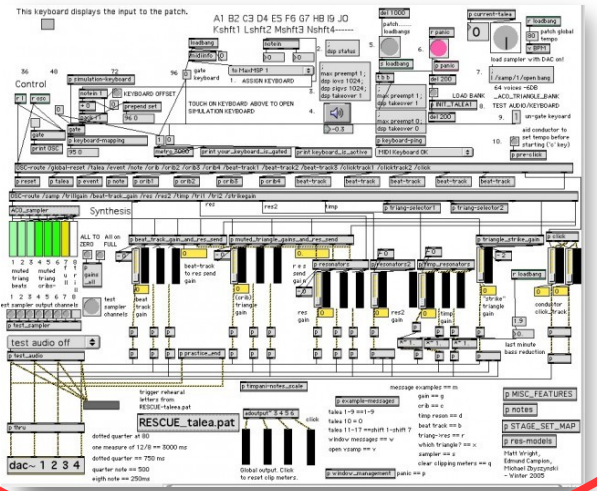
macros



scripting

```
AddEntityCollideCallback("Player","AreaHelp",
   "CollidePlayerWithAreaHelp");

void CollidePlayerWithAreaHelp()
{
    if(HasItem("cellar_key") == true)
    {
        SetMessage("Use your inventory to select "+
        "key and use on door.", 10);
    }
    else
    {
        SetMessage("You will need a key to open "+
        "the door.", 5);
    }

}
```

node editors



Scripting

"little languages"

eqn

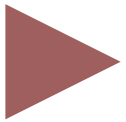
pic sed awk

SCUMM

Scripting

"little languages"

eqn
pic sed awk
SCUMM



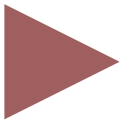
custom DSLs

GraForth
WordBasic
AutoLISP

Scripting

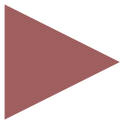
"little languages"

eqn
pic sed awk
SCUMM



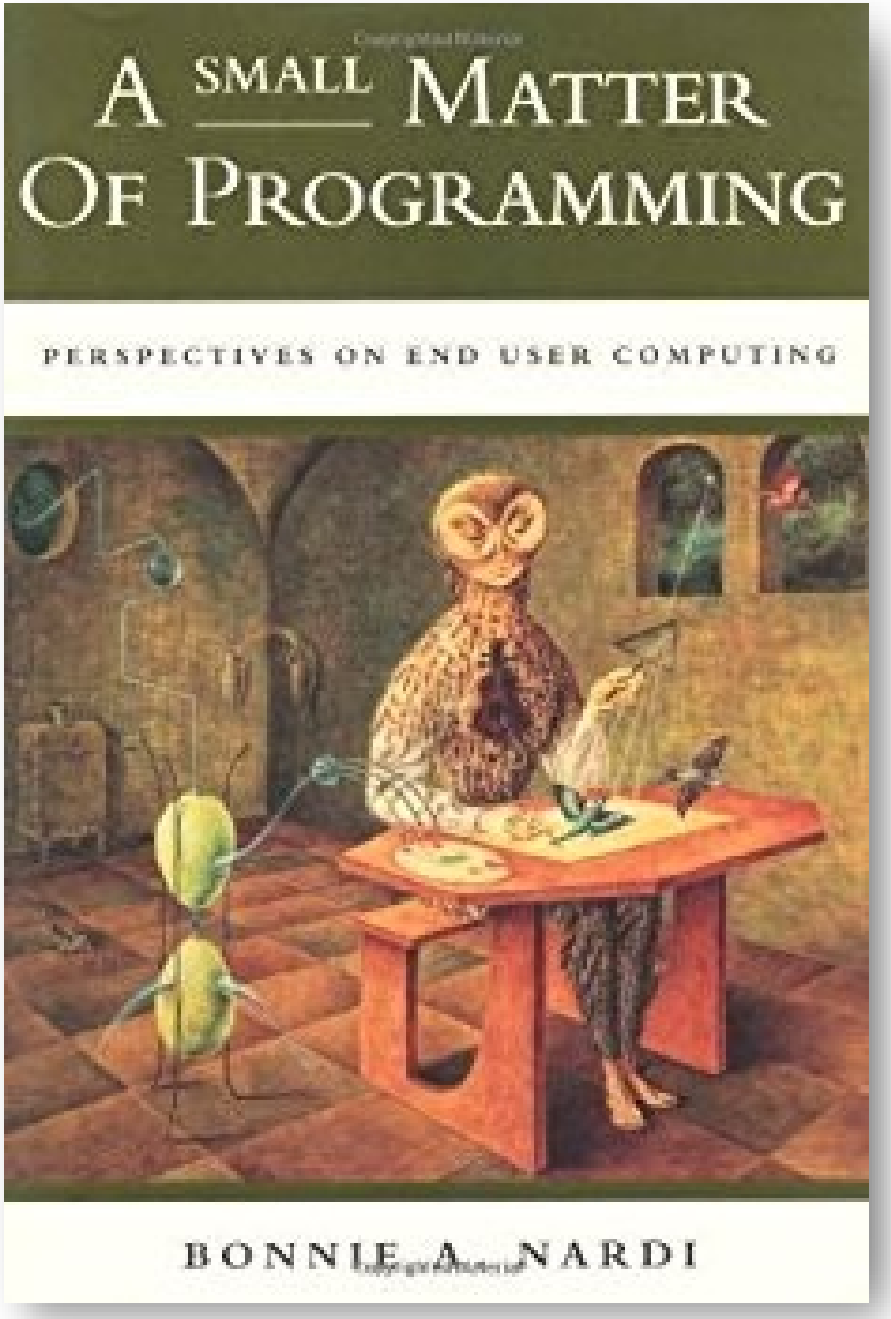
custom DSLs

GraForth
WordBasic
AutoLISP



scripting languages

Tcl Ruby
Lua
Python


A SMALL MATTER OF PROGRAMMING
PERSPECTIVES ON END USER COMPUTING
BONNIE A. NARDI

Users and developers

End-users

- Domain specialists
- Care about the domain, not the software

Domain developers

- Domain specialists interested in the software
- Advanced users: “Tinkerers”, “enthusiasts”

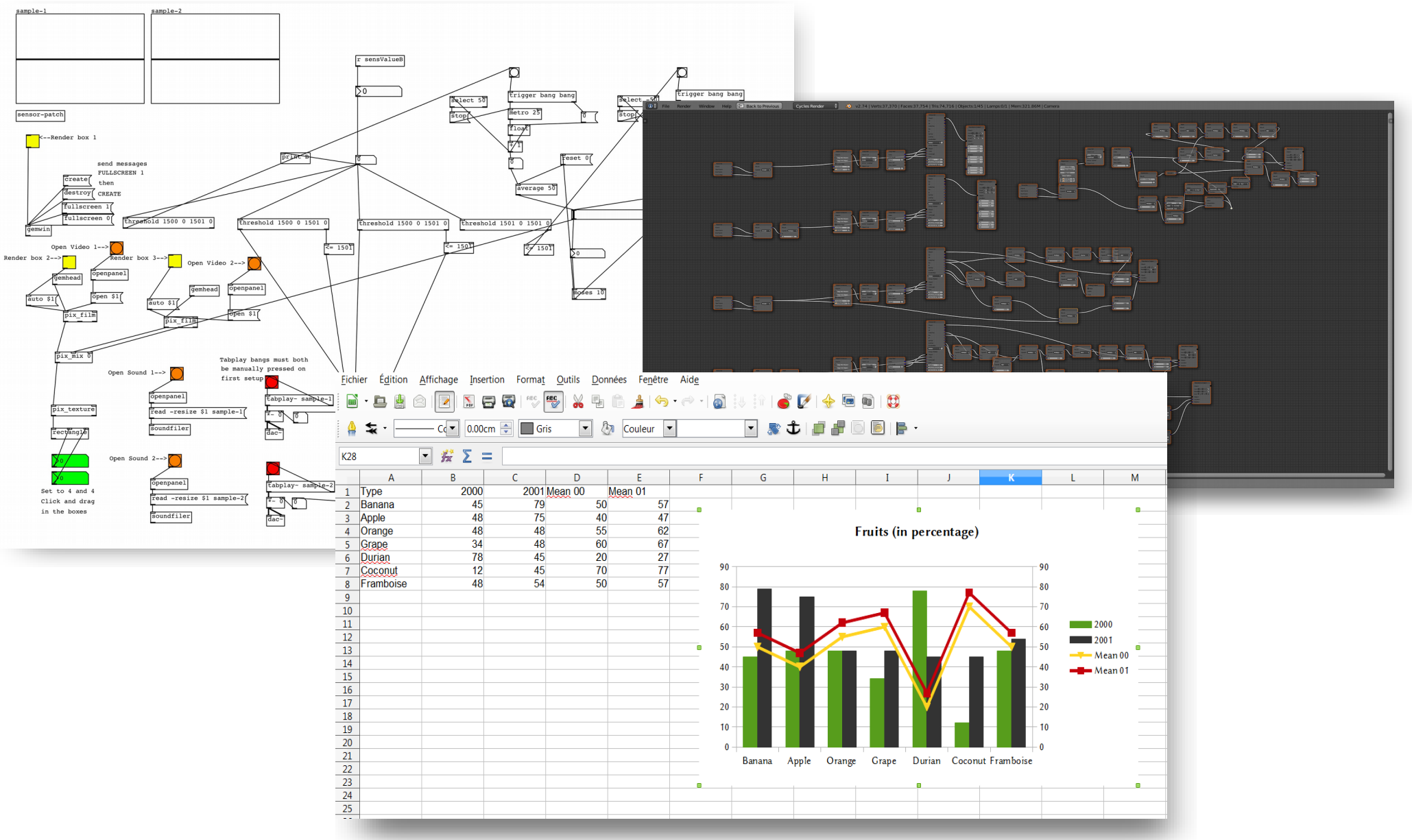
Developers

- Software development professionals

Users and architectural layers

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

UI-Level languages



Dataflow

UI-Level Languages

ad-hoc languages

Pure Data
Blender
Excel
LabVIEW
VEE
Reaktor

UI-Level Languages

ad-hoc languages

?

Pure Data

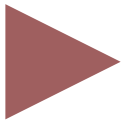
Blender

Excel

LabVIEW

VEE

Reaktor



? ? ? ? ? ?

How should UI-level languages evolve?

When designing a dataflow language
for an end-user programmable application,
which design choices
should be taken into consideration
with regard to its semantics
and what are their effects?

When designing a dataflow language
for an end-user programmable application,
which design choices
should be taken into consideration
with regard to its semantics
and what are their effects?

When designing a dataflow language
for an end-user programmable application,
which design choices
should be taken into consideration
with regard to its semantics
and what are their effects?

When designing a dataflow language
for an end-user programmable application,
which design choices
should be taken into consideration
with regard to its semantics
and what are their effects?

When designing a dataflow language
for an end-user programmable application,
which design choices
should be taken into consideration
with regard to its semantics
and what are their effects?

When designing a dataflow language
for an end-user programmable application,
which design choices
should be taken into consideration
with regard to its semantics
and what are their effects?

Case studies

of dataflow end-user programmable applications

When designing a dataflow language
for an end-user programmable application,
which design choices
should be taken into consideration
with regard to its semantics
and what are their effects?

Case studies

of dataflow end-user programmable applications

Design alternatives

pertaining their semantics

When designing a dataflow language
for an end-user programmable application,
which design choices
should be taken into consideration
with regard to its semantics
and what are their effects?

Case studies

of dataflow end-user programmable applications

Design alternatives

pertaining their semantics

Critique

of the effects of these design choices

Case studies

Case studies

- Understanding the design of *real-world* dataflow UI-level languages
- *Definitional interpreters* in Haskell, written in the style of operational semantics
- *Analyze* their differences and their common patterns

Three in-depth studies

- Pure Data
- Spreadsheet formula language
- LabVIEW

Three in-depth studies

- Pure Data

  Chapter 4

- Spreadsheet formula language

  Chapter 5

- LabVIEW

  Chapter 6

Three in-depth studies

- Pure Data

  Chapter 4

  http://github.com/hishamhm/puredata

- Spreadsheet formula language

  Chapter 5

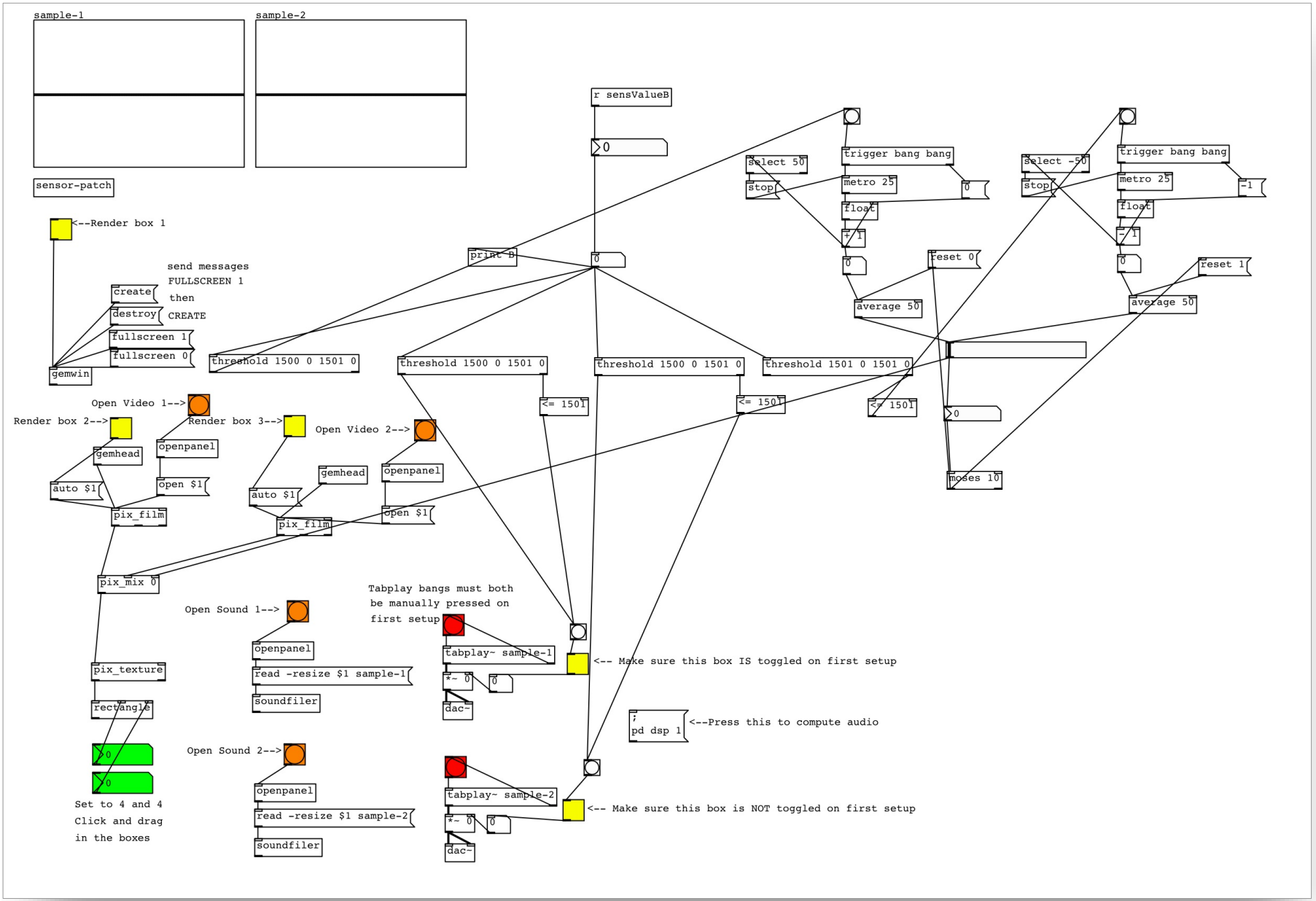
  http://github.com/hishamhm/spreadsheet

- LabVIEW

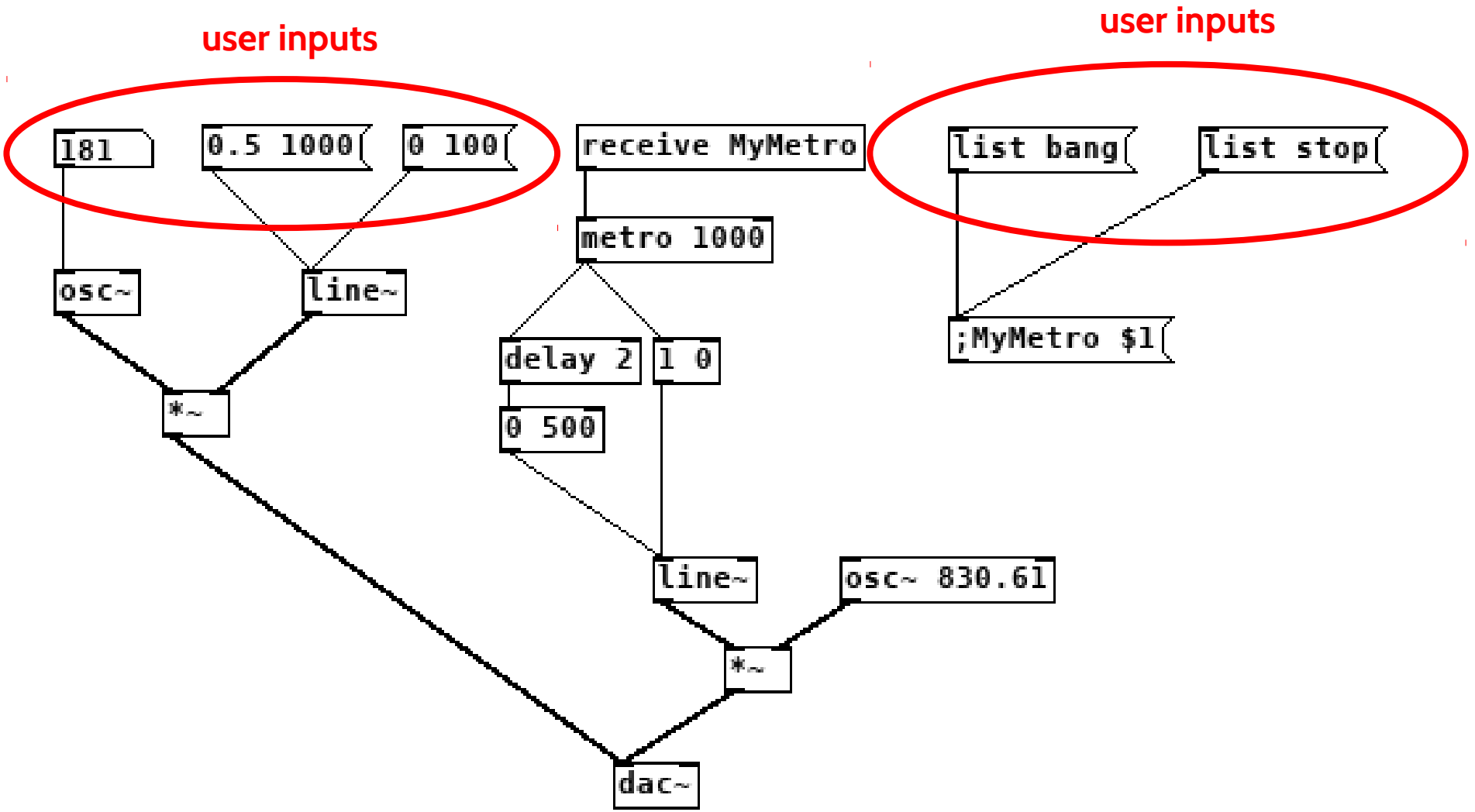
  Chapter 6

  http://github.com/hishamhm/lv

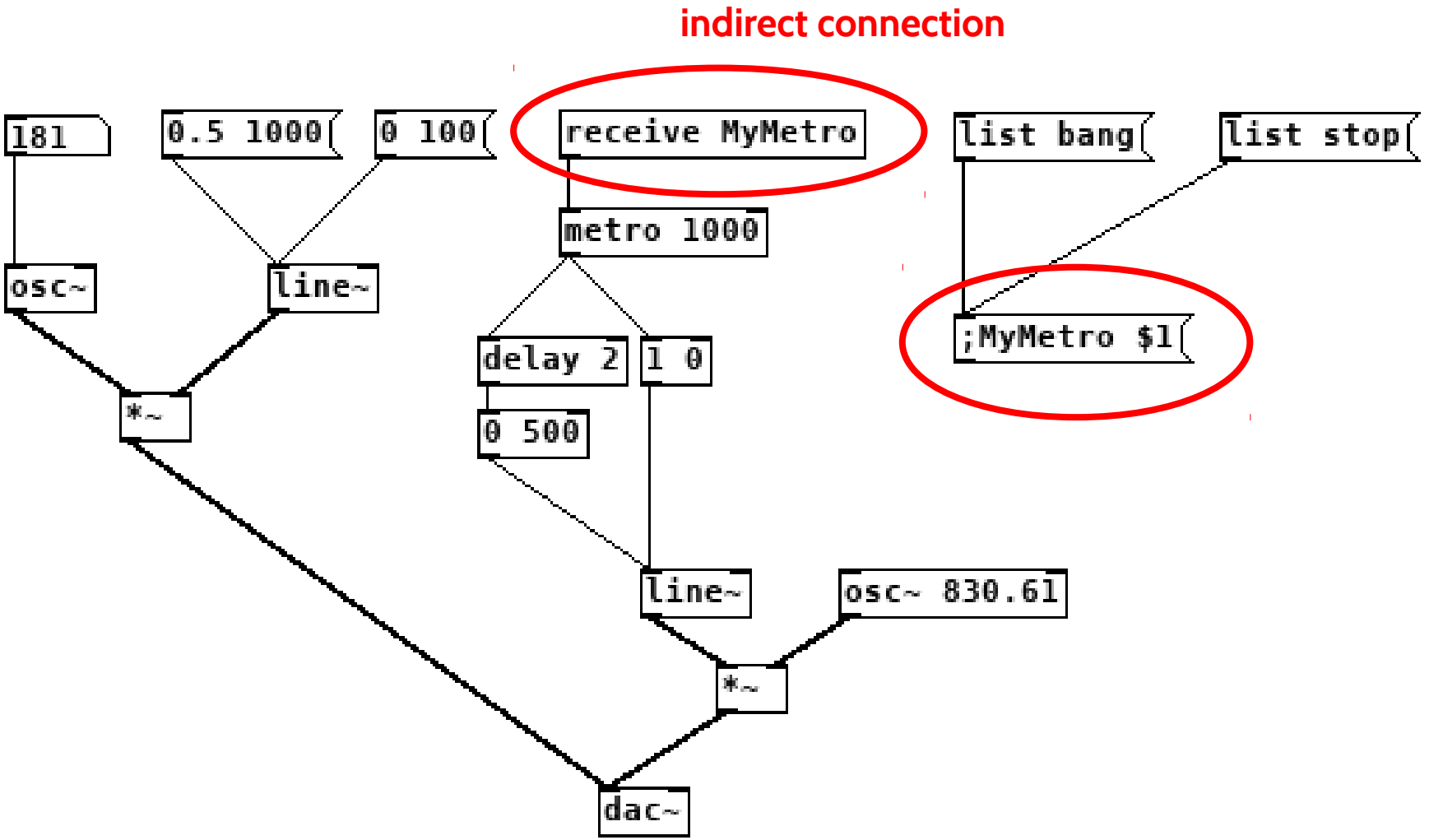
Pure Data


sample-1
sample-2
sensor-patch
<--Render box 1
send messages
FULLSCREEN 1
then
CREATE
create
destroy
fullscreen 1
fullscreen 0
gemwin
threshold 1500 0 1501 0
print B
r sensValueB
0
select 50
stop
trigger bang bang
metro 25
float
+ 1
average 50
reset 0
select -50
-1
reset 1
<= 1501
threshold 1501 0 1501 0
moses 10
Open Video 1-->
Render box 2-->
Render box 3-->
Open Video 2-->
gemhead
openpanel
auto $1
open $1
pix_film
pix_mix 0
Tabplay bangs must both
be manually pressed on
first setup
Open Sound 1-->
tabplay~ sample-1
<-- Make sure this box IS toggled on first setup
*~ 0
dac~
pix_texture
rectangle
Open Sound 2-->
read -resize $1 sample-1
soundfiler
; pd dsp 1
<--Press this to compute audio
tabplay~ sample-2
<-- Make sure this box is NOT toggled on first setup
read -resize $1 sample-2
Set to 4 and 4
Click and drag
in the boxes

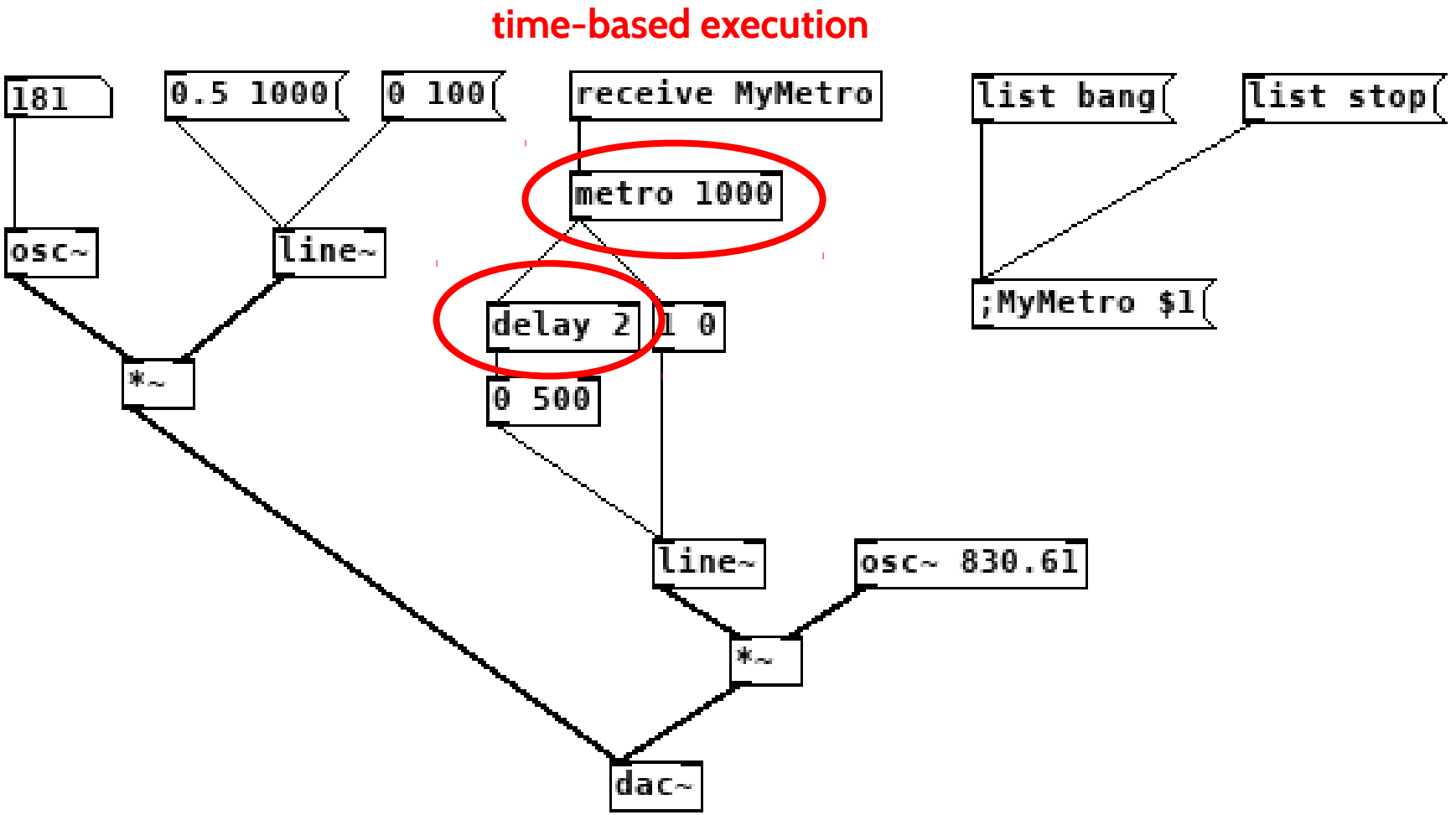
An example


user inputs
user inputs
181
0.5 1000
0 100
receive MyMetro
list bang
list stop
metro 1000
osc~
line~
;MyMetro $1
delay 2
1 0
*~
0 500
line~
osc~ 830.61
*~
dac~

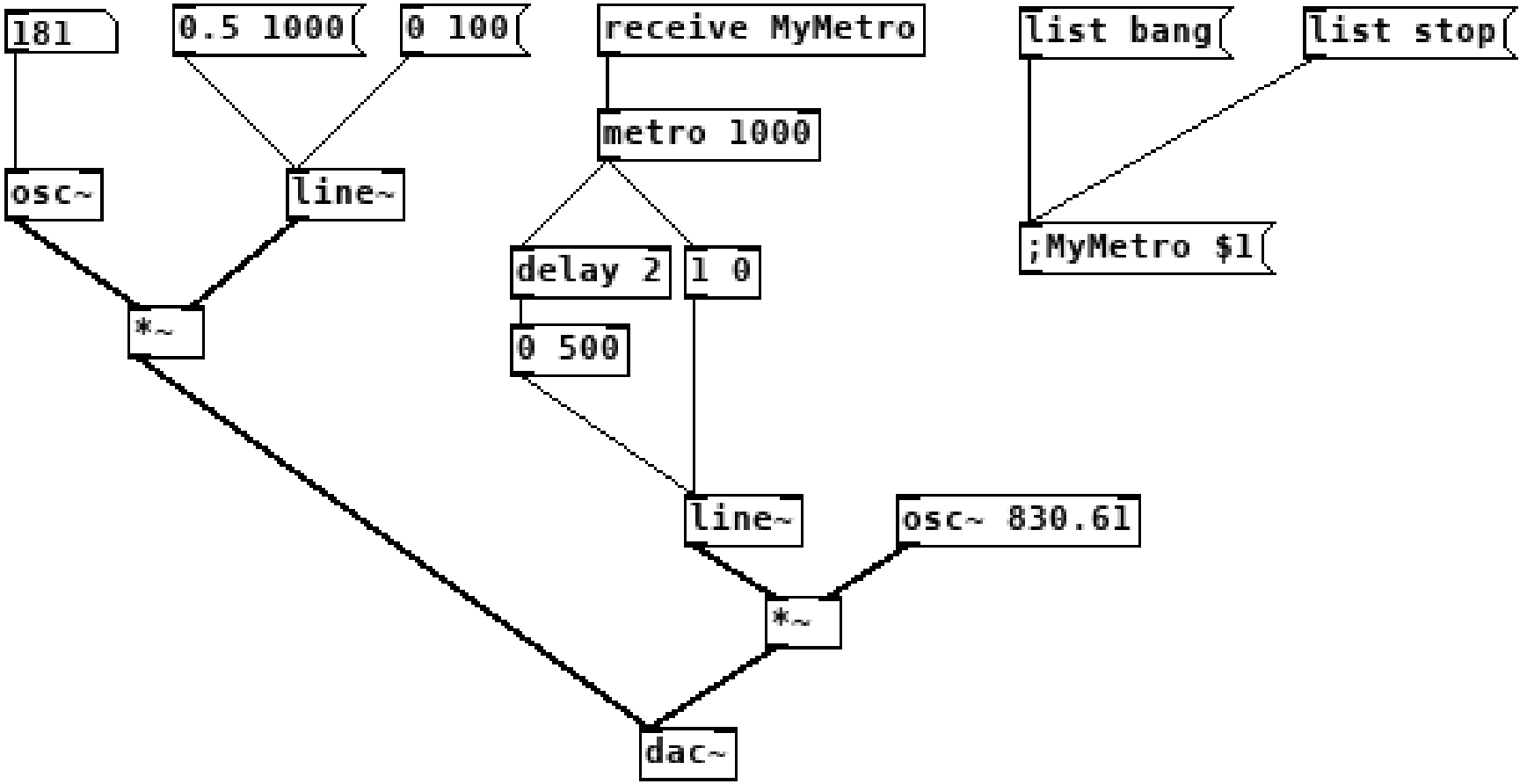
An example


indirect connection
181
0.5 1000
0 100
receive MyMetro
list bang
list stop
metro 1000
osc~
line~
delay 2
1 0
;MyMetro $1
*~
0 500
line~
osc~ 830.61
*~
dac~

An example


time-based execution
181
0.5 1000
0 100
receive MyMetro
list bang
list stop
metro 1000
osc~
line~
;MyMetro $1
delay 2
1 0
*~
0 500
line~
osc~ 830.61
*~
dac~

A demo!

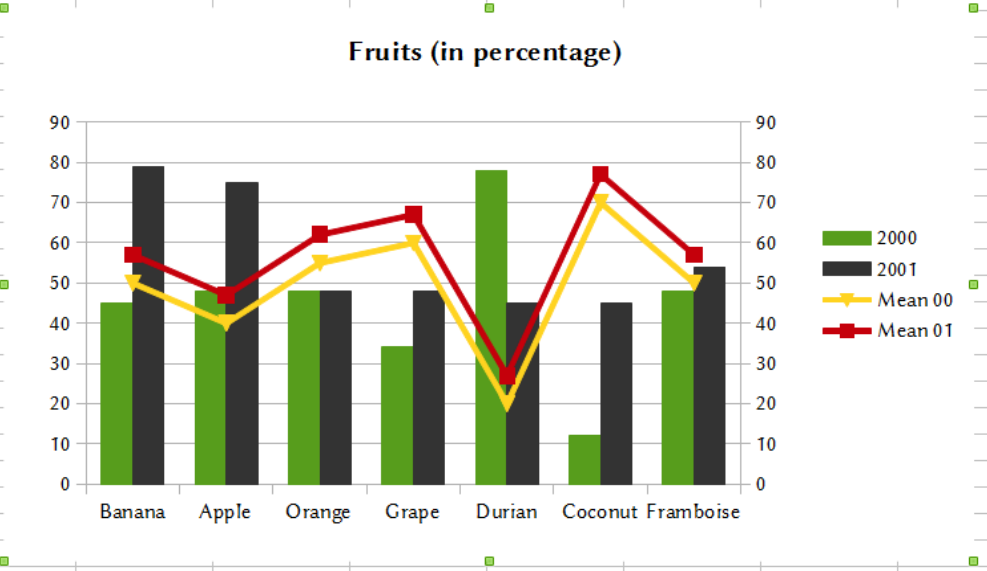

181
0.5 1000
0 100
receive MyMetro
list bang
list stop
metro 1000
osc~
line~
delay 2
1 0
;MyMetro $1
*~
0 500
line~
osc~ 830.61
*~
dac~

Spreadsheets

Fichier Édition Affichage Insertion Format Outils Données Fenêtre Aide

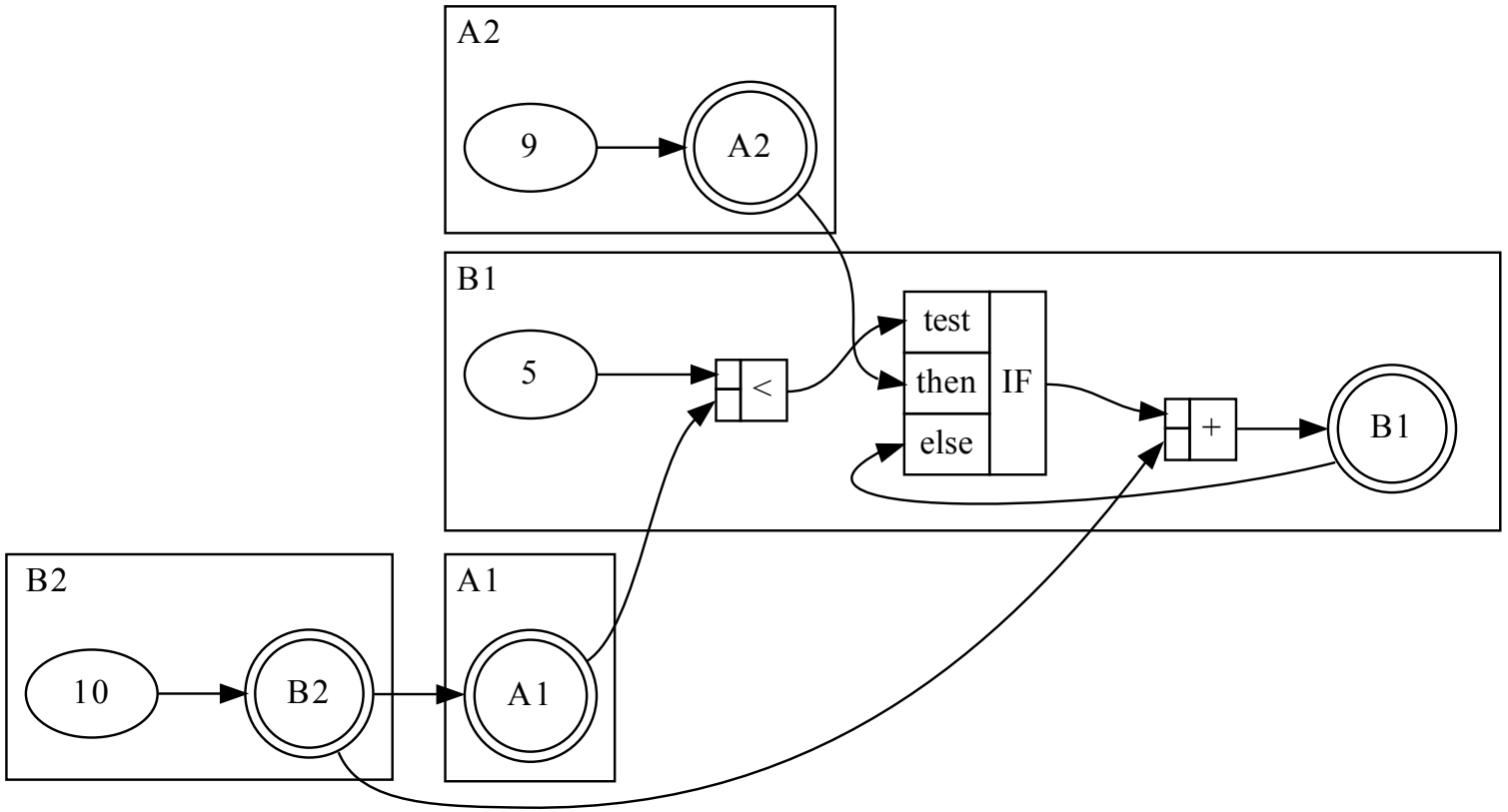
K28

	A	B	C	D	E
1	Type	2000	2001	Mean 00	Mean 01
2	Banana	45	79	50	57
3	Apple	48	75	40	47
4	Orange	48	48	55	62
5	Grape	34	48	60	67
6	Durian	78	45	20	27
7	Coconut	12	45	70	77
8	Framboise	48	54	50	57



Spreadsheets as dataflow

	A	B
1	=B2	=IF(5<A1;A2:B1)+B2
2	9	10



Which spreadsheet?

Microsoft Excel 2010

LibreOffice Calc 5

Google Sheets

Microsoft Excel Online

Microsoft Excel for Android

Incompatibilities

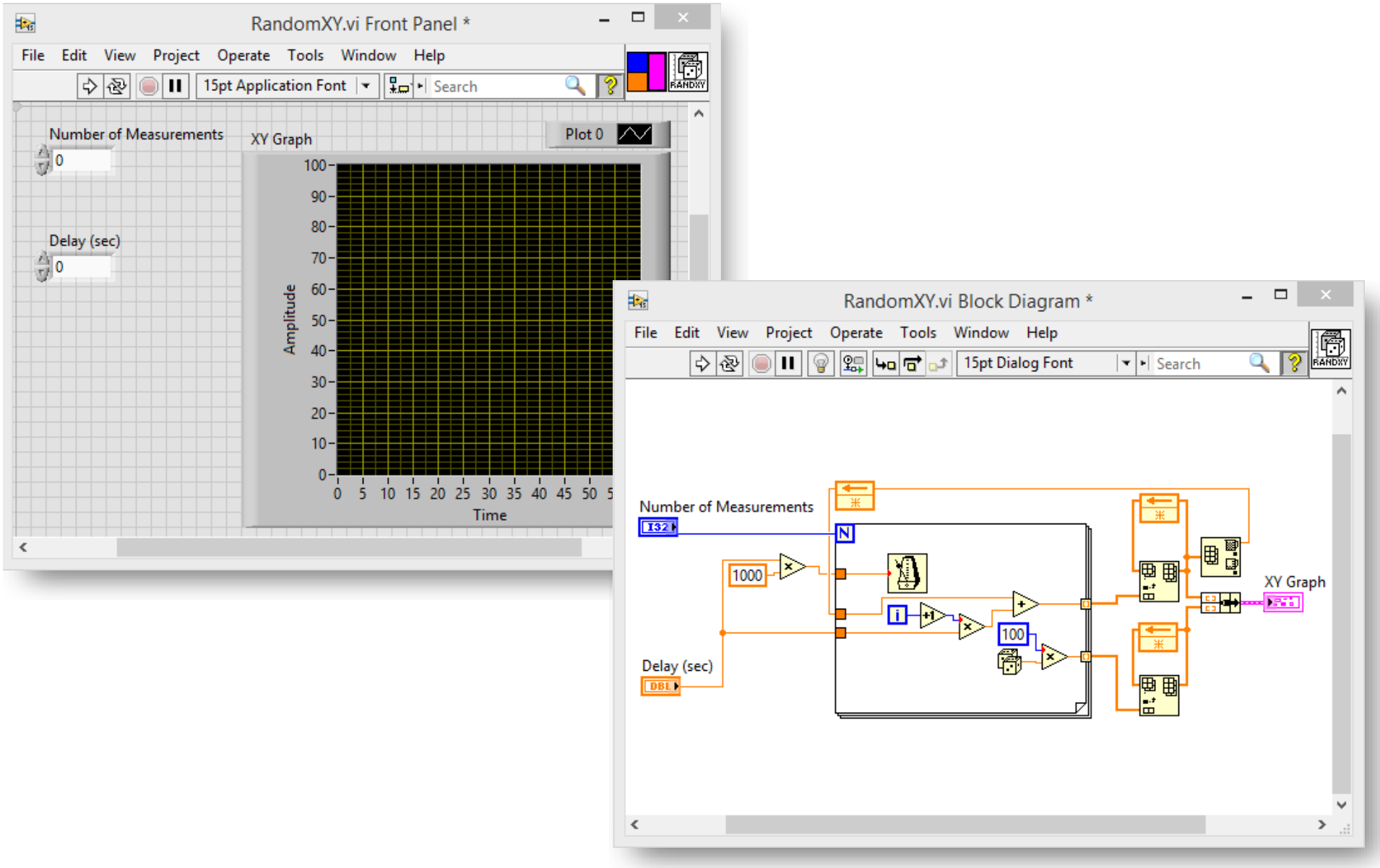
		Microsoft Excel	LO Calc	Google Sheets	Excel Online
=SUM(SQRT({10,20}))	F	7.6344	3.1622	3.1622	7.6344
	AF	7.6344	7.6344	7.6344	
=SUM(SQRT(A1:A2))	F	#VALUE!	#VALUE!	#VALUE!	#VALUE!
	AF	7.6344	7.6344	7.6344	
=SUM(SQRT(INDIRECT({"A1","A2"}))	F	#VALUE!	3.1622	3.1622	#VALUE!
	AF	#VALUE!	7.6344	3.1622	
=SUM(INDIRECT({"A1","A2"})	F	10	10	10	10
	AF	10	30	10	
=SUM(MINVERSE(A1:B2))	F	4.163E-17	27756E-17	0	#VALUE!

The spreadsheet interpreter

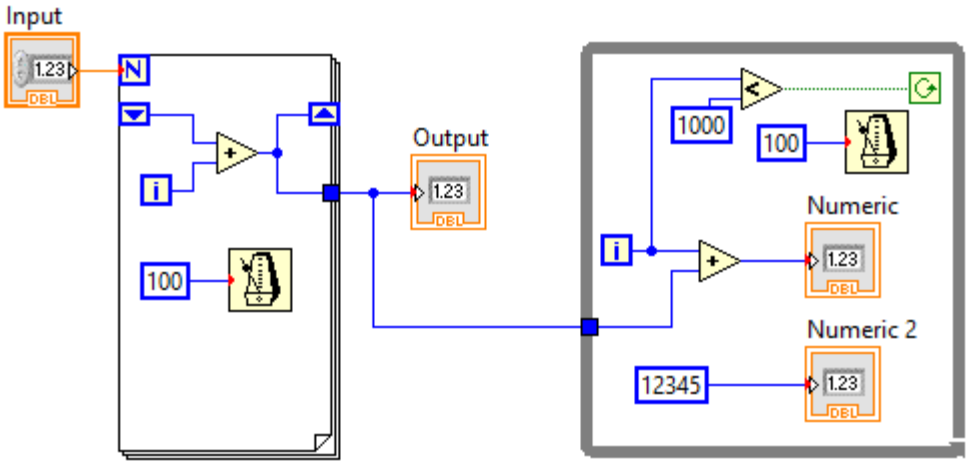
	A	B	C	D	E	F	G
1	15	15	"B"	75	30	105	1015
2	30	15	1				
3							
4							
5			"#VALUE!"	115		15	
6				130		16	
7							
8							"#VALUE!"
9							
10	10			10	-20	30	
11	"10"				20		
12	False	"#DIV/0!"					
13	True	"#VALUE!"					
14	True	"#DIV/0!"					

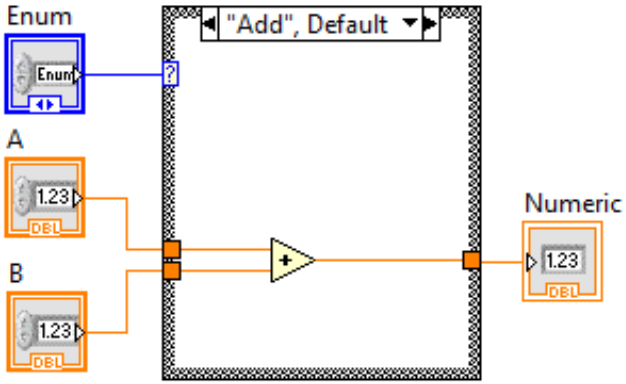
Tests against the ISO and OpenDocument standards in
http://github.com/hishamhm/spreadsheet

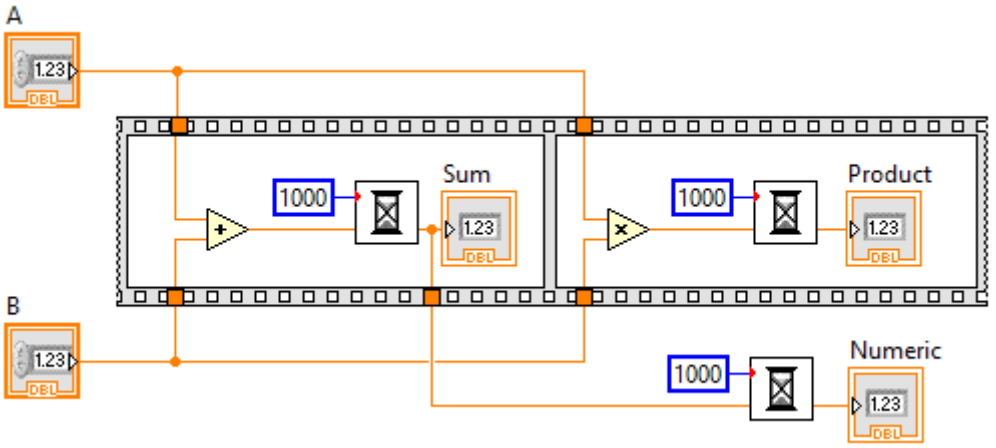
LabVIEW


RandomXY.vi Front Panel *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Number of Measurements
0
Delay (sec)
0
XY Graph
Plot 0
Amplitude
Time
RandomXY.vi Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Dialog Font
Search
Number of Measurements
I32
Delay (sec)
DBL
1000
100
XY Graph

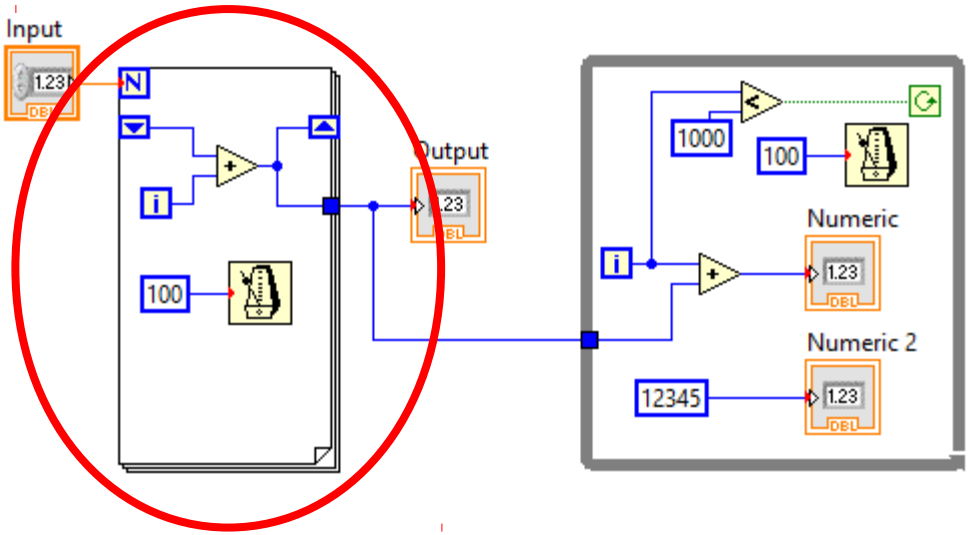
Control structures



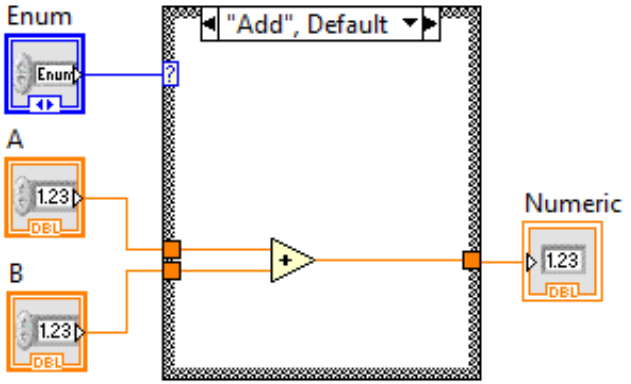


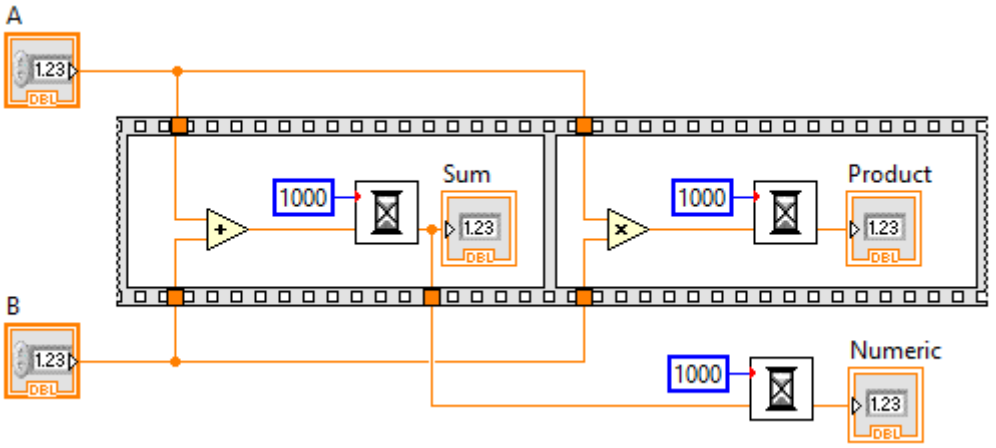


Control structures

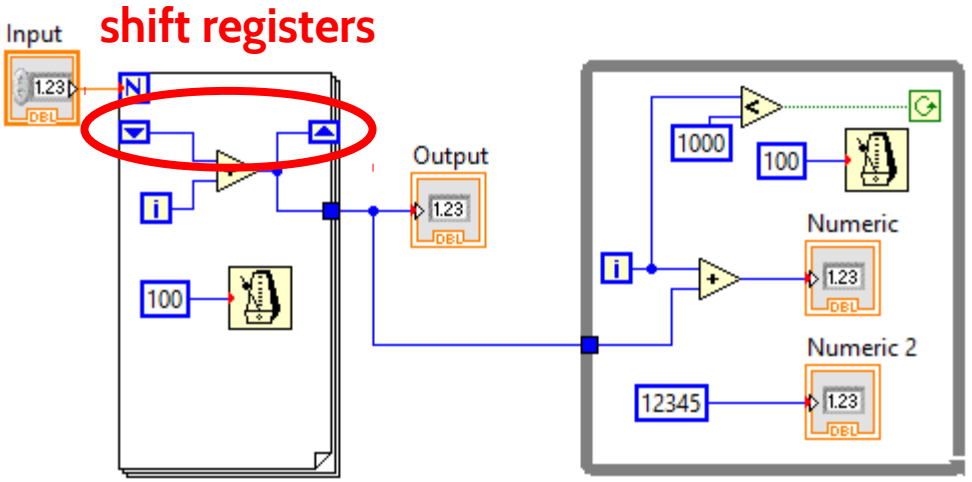


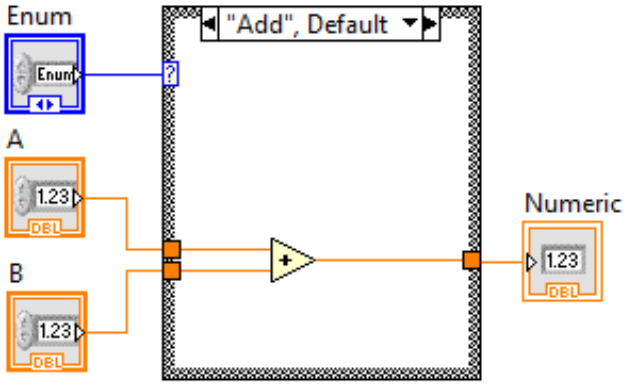
for loop

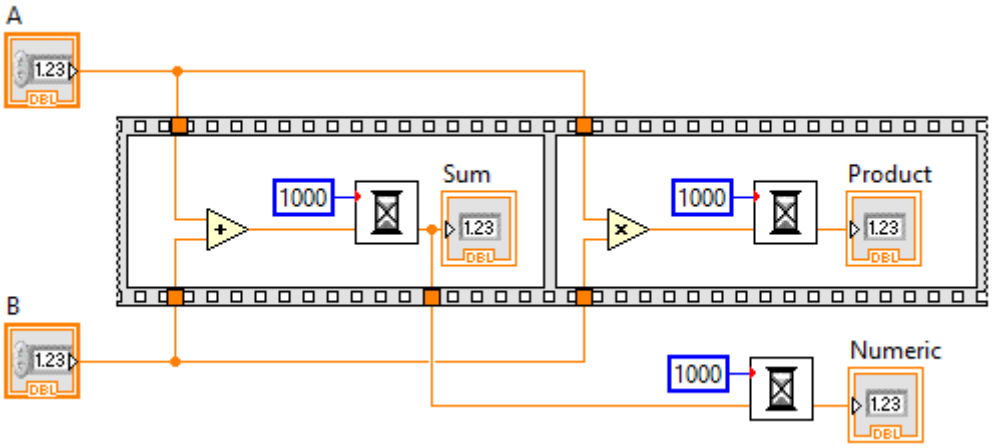




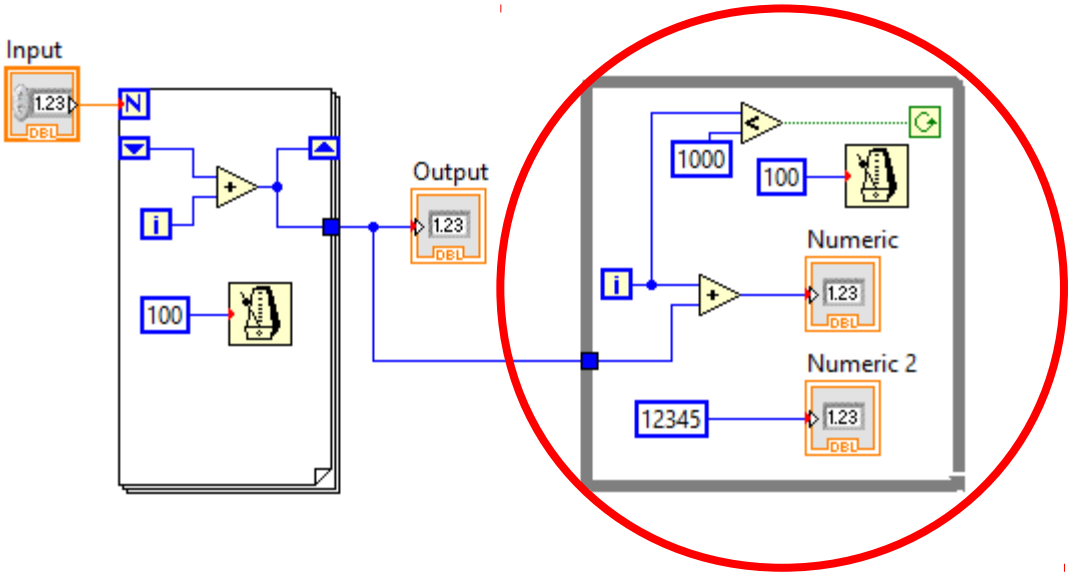
Control structures



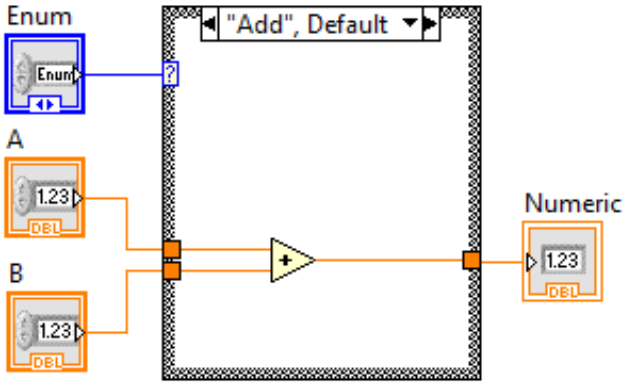


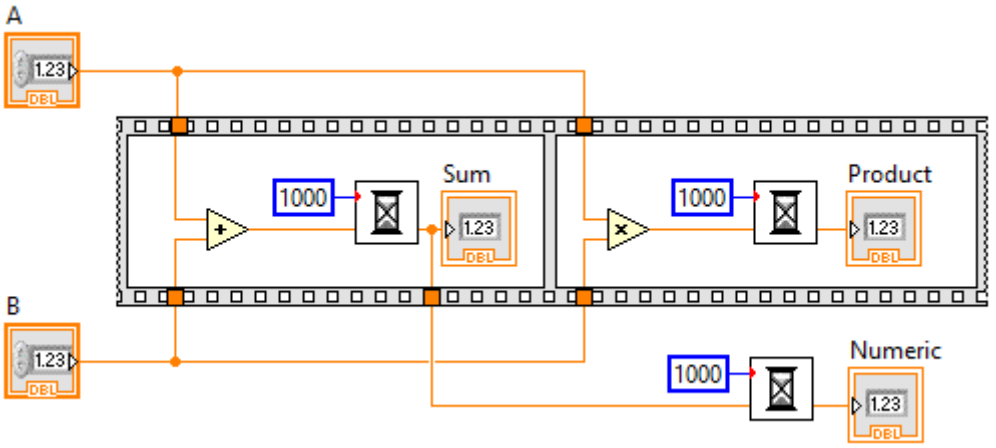


Control structures

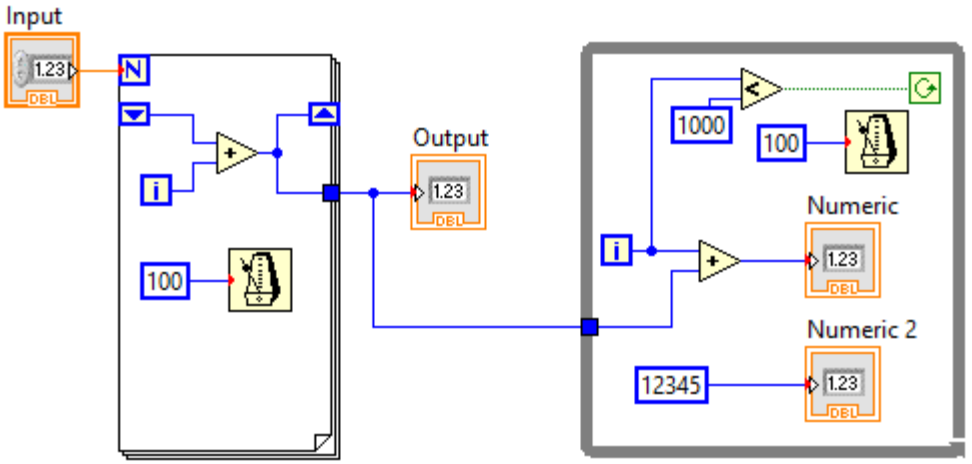


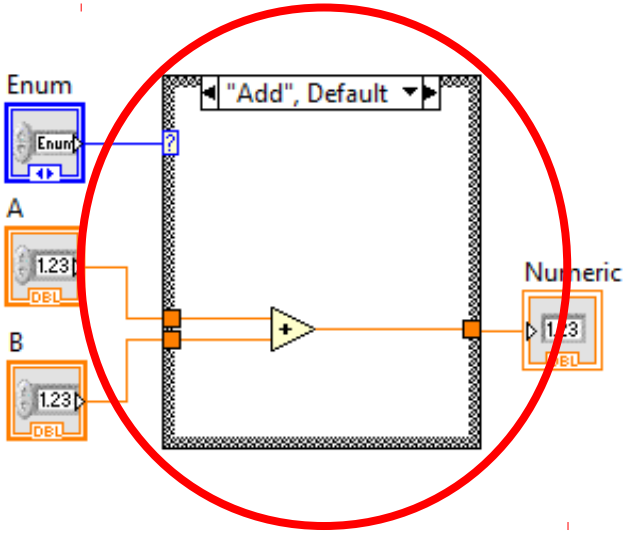
while loop



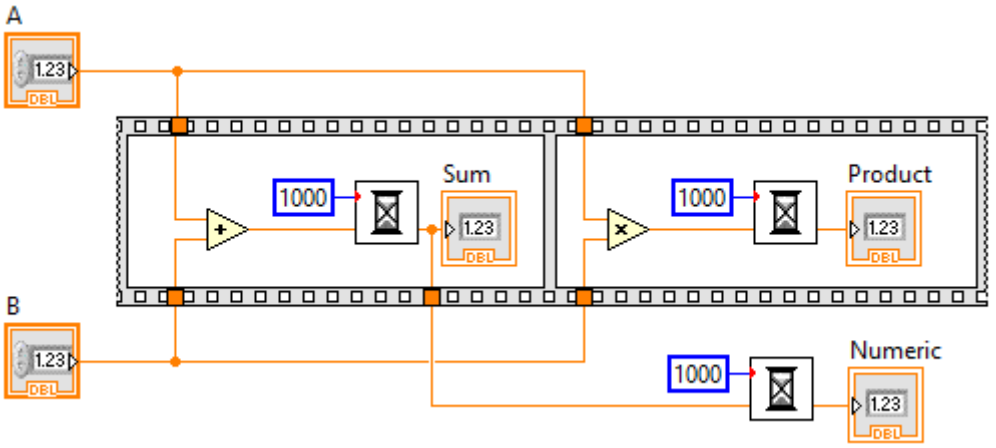


Control structures

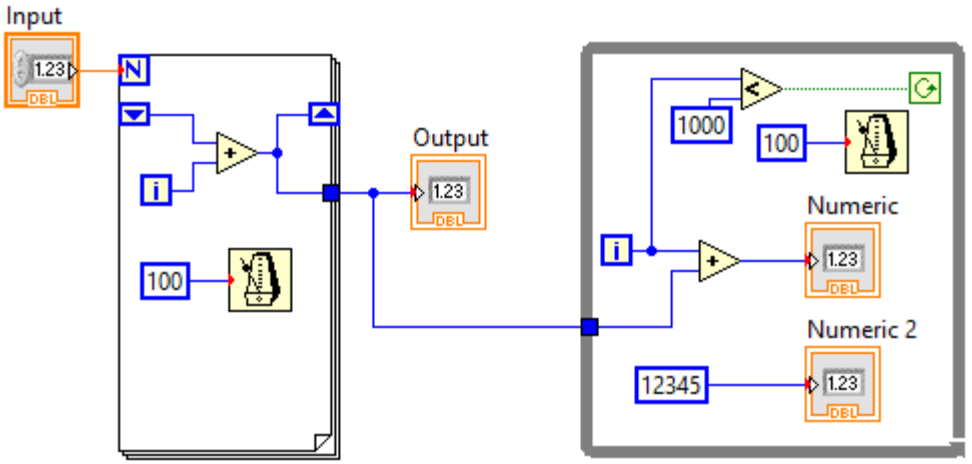


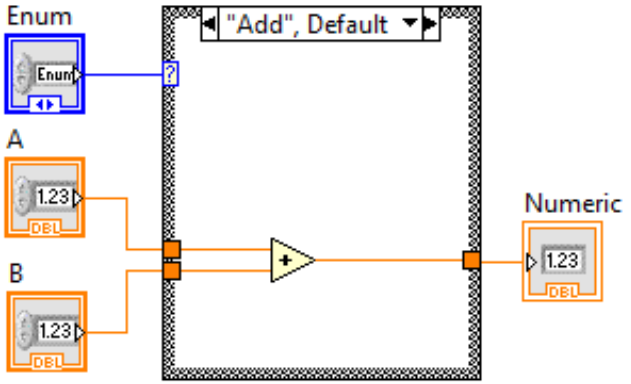


case structure

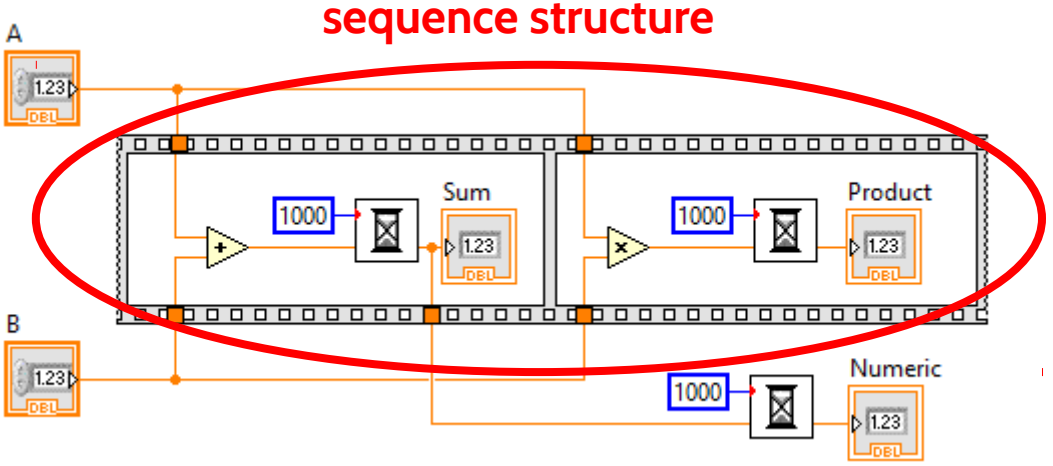


Control structures

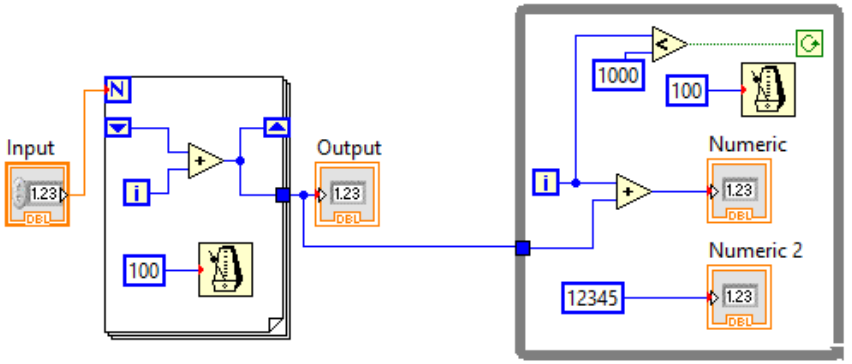


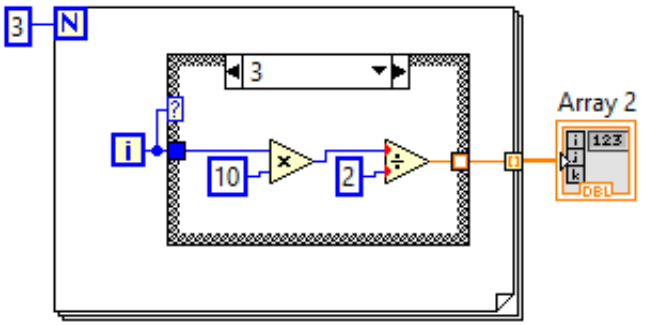


sequence structure



Some demos!


Input
Output
1.23
DBL
N
i
100
1000
Numeric
Numeric 2
12345


3
N
i
10
2
Array 2
123
DBL

Design alternatives

Journal of Visual Languages and Computing (1992) 3, 69–101

Visual Languages and Computing Survey: Data Flow Visual Programming Languages

DANIEL D. HILS

Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, Illinois 61801, U.S.A.

Received 20 November 1990 and accepted 25 May 1991

The data flow model is a popular model on which to base a visual programming language. This paper describes alternatives available to a designer of data flow languages, describes many of the languages, discusses some strengths of the languages, and discusses some unsolved problems in the design of data flow languages.

1. Introduction

DATA FLOW IS A POPULAR COMPUTATIONAL MODEL for visual programming languages. Data flow provides a view of computation which shows the data flowing from one filter function to another, being transformed as it goes. In addition, the data flow model easily accomodates the insertion of viewing monitors at various points to show the data to the user. Consequently, many recent visual programming languages are based on the data flow model.

This paper describes many of the data flow visual programming languages. The languages are grouped according to their application domain. For each language, pertinent aspects of its appearance, and the particular design alternatives it uses, are discussed. Next, some strengths of data flow visual programming languages are mentioned. Finally, unsolved problems in the design of such languages are discussed.

2. Methods for Classification of Data Flow Languages

Data flow visual programming languages may be classified according to their use of various design alternatives or according to their application domain. This paper groups languages by their application domain, and mentions the design alternatives which each language uses. Table 1 summarizes the design alternatives used by various languages.

2.1. Design Alternatives

The idea of using a data flow graph to represent a program has existed for some time [1]. However, the widespread use of the data flow computational model in visual programming languages is more recent [2]. The central concept of the data flow model is that a program can be represented by a directed graph where nodes represent functions and where arcs represent the flow of data between functions [3]. Arcs going into a node represent input data to a function; arcs going out represent output data—that is, the function's results. Units of data which flow on the arcs are called tokens; arcs may also be called links. Throughout this paper, the terms *arc*, *wire*, *link* and *line* are used interchangeably, as are *procedure*, *function*, *icon* and *box*.

1045-926X/92/010069 + 33 $03.00/0

Hils' design alternatives

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)

An extension to this categorization

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)

Excercising the categorization

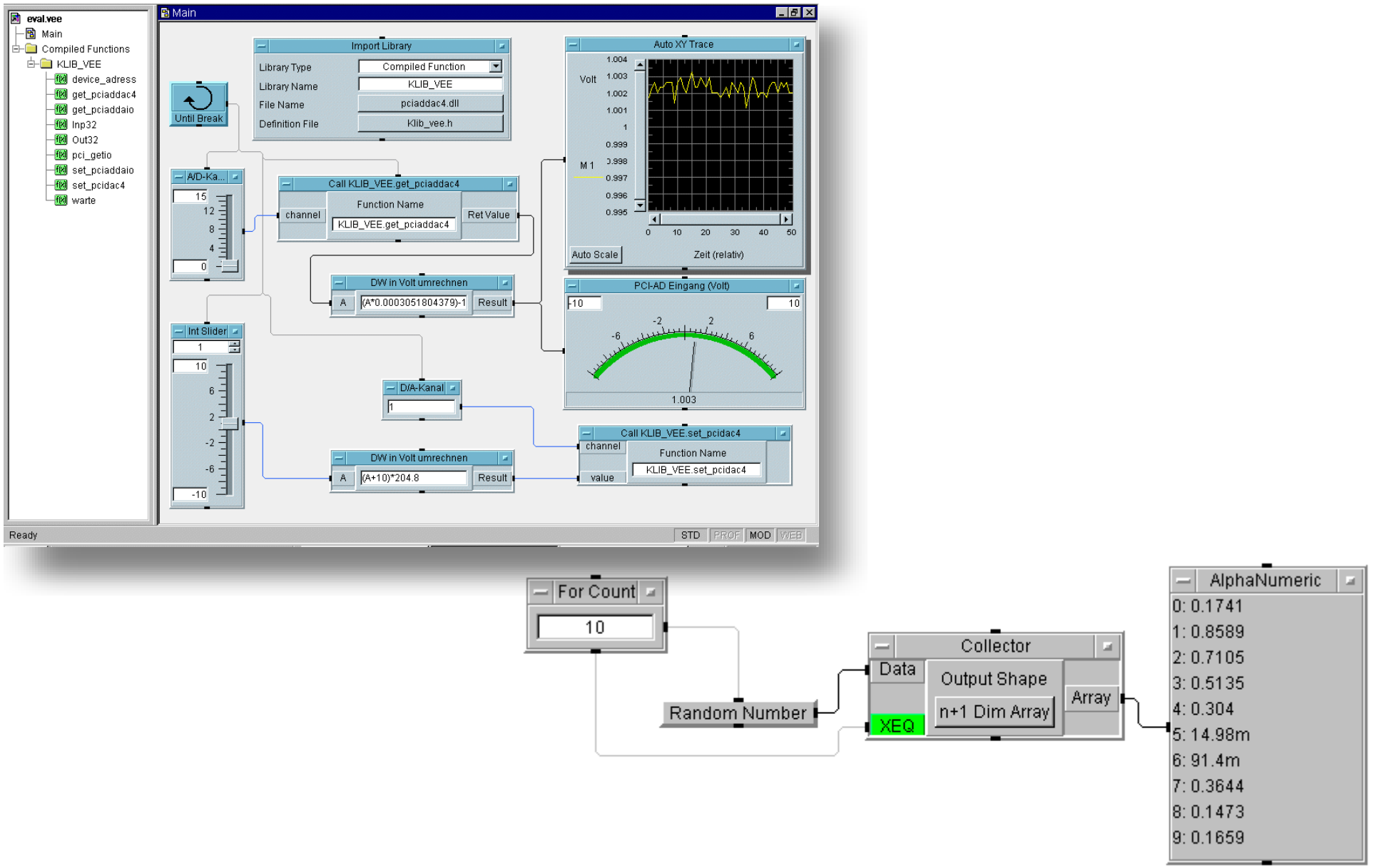
Three additional languages:

VEE

Blender

Reaktor

VEE


eval.vee
Main
Compiled Functions
KLIB_VEE
device_adress
get_pciaddac4
get_pciaddaio
Inp32
Out32
pci_getio
set_pciaddaio
set_pcidac4
warte
Main
Import Library
Library Type
Compiled Function
Library Name
KLIB_VEE
File Name
pciaddac4.dll
Definition File
Klib_vee.h
Until Break
Auto XY Trace
Volt
M 1
Auto Scale
Zeit (relativ)
A/D-Ka...
Call KLIB_VEE.get_pciaddac4
channel
Function Name
KLIB_VEE.get_pciaddac4
Ret Value
DW in Volt umrechnen
A
(A*0.0003051804379)-1
Result
PCI-AD Eingang (Volt)
Int Slider
D/A-Kanal
Call KLIB_VEE.set_pcidac4
value
DW in Volt umrechnen
(A+10)*204.8
Ready
STD
PROF
MOD
WEB
For Count
10
Random Number
Collector
Data
Output Shape
n+1 Dim Array
Array
XEQ
AlphaNumeric
0: 0.1741
1: 0.8589
2: 0.7105
3: 0.5135
4: 0.304
5: 14.98m
6: 91.4m
7: 0.3644
8: 0.1473
9: 0.1659

Blender

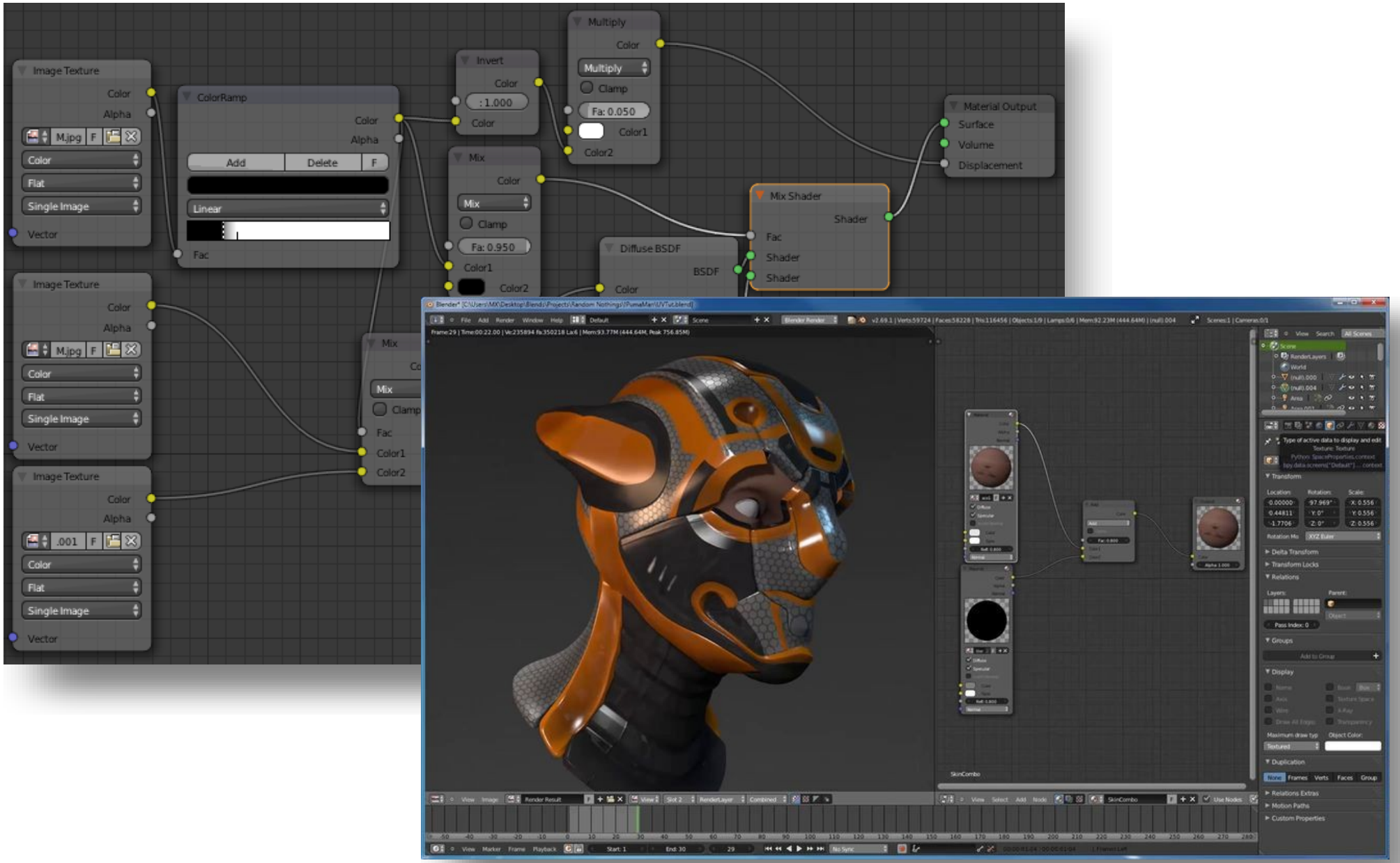


Image Texture
Color
Alpha
M.jpg
F
Color
Flat
Single Image
Vector
ColorRamp
Color
Alpha
Add
Delete
F
Linear
Fac
Invert
Color
: 1.000
Color
Multiply
Color
Multiply
Clamp
Fa: 0.050
Color1
Color2
Material Output
Surface
Volume
Displacement
Mix
Color
Mix
Clamp
Fa: 0.950
Color1
Color2
Mix Shader
Shader
Fac
Shader
Shader
Diffuse BSDF
BSDF
Color
Image Texture
Color
Alpha
M.jpg
F
Color
Flat
Single Image
Vector
Mix
Mix
Clamp
Fac
Color1
Color2
Image Texture
Color
Alpha
.001
F
Color
Flat
Single Image
Vector

Reaktor


8STEPS
LFO
OSC
COMB
ENV
FILTER
DELAY
REVERB
PAUL
Convert
Event Proc
Flow
Logic
Math
Math Mod
Memory
Saturator
Primary
Reaktor Blocks
1 div sqrt.rcm
1 div x.rcm
1 wrap.rcm
IDivMod.rcm
IMax.rcm
IMin.rcm
Routing[]
Delay 1P 1083N
Macro
to next & delay

Critique

General information	**Pure Data**	**Excel**	**LabVIEW**	**Reaktor**	**VEE**	**Blender**
Main reference	[P+15]		[Nat01]	[Nat15]	[Agi11]	[Ble17]
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

Commonalities in UI-Level languages

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

Commonalities in UI-Level languages

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

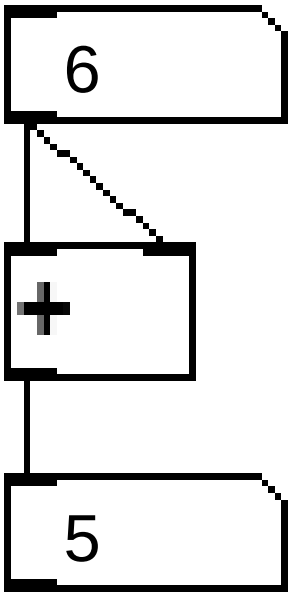
Feature interactions

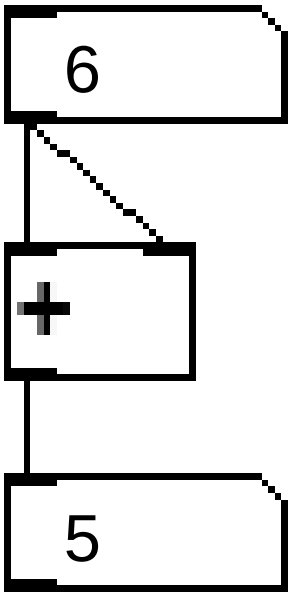
Design alternatives [Hil92]	Pure Data	Excel	LabVIEW	Reaktor	VEE	Blender
Box-line representation	Yes	No	Yes	Yes	Yes	Yes
Iteration	Yes (cycles)	Limited	Yes (construct)	Limited	Yes	No
Subprogram abstraction	Yes	No	Yes	Yes	Yes	Yes
Selector/distributor	Yes	Yes	Yes	Yes	Yes	Yes
Flow of data	Uni	Uni	Uni	Uni	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

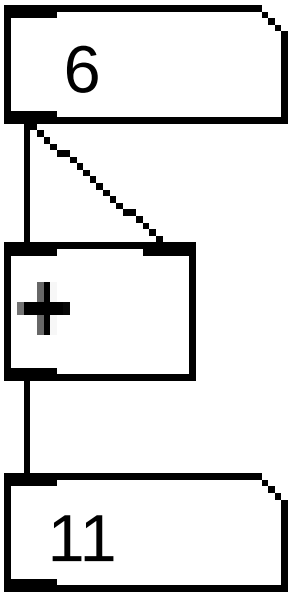
Additional design alternatives	Pure Data	Excel	LabVIEW	Reaktor	VEE	Blender
Dataflow model	Dynamic	Static	Static	Static	Static	Static
N-to-1 inputs	Yes	No	No	No	No	No
Separate edit/use views	No	No	Yes	Yes	Yes	No
Time-dependent firing	Yes	No	Yes	Yes	Yes	No
Rate-based evaluation	Synchronous	No	No	Synchronous	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

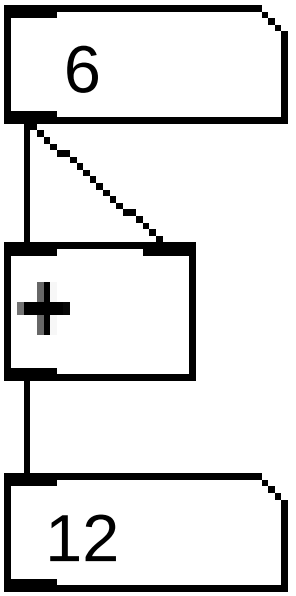
Feature interaction in Pure Data

- Dynamic dataflow model
  - graph loops for building delays and echoes
  - triggering and non-triggering inputs
- N-to-1 inputs in a port
- Naive model for deterministic execution


6
+
5


6
+
5


6
+
11

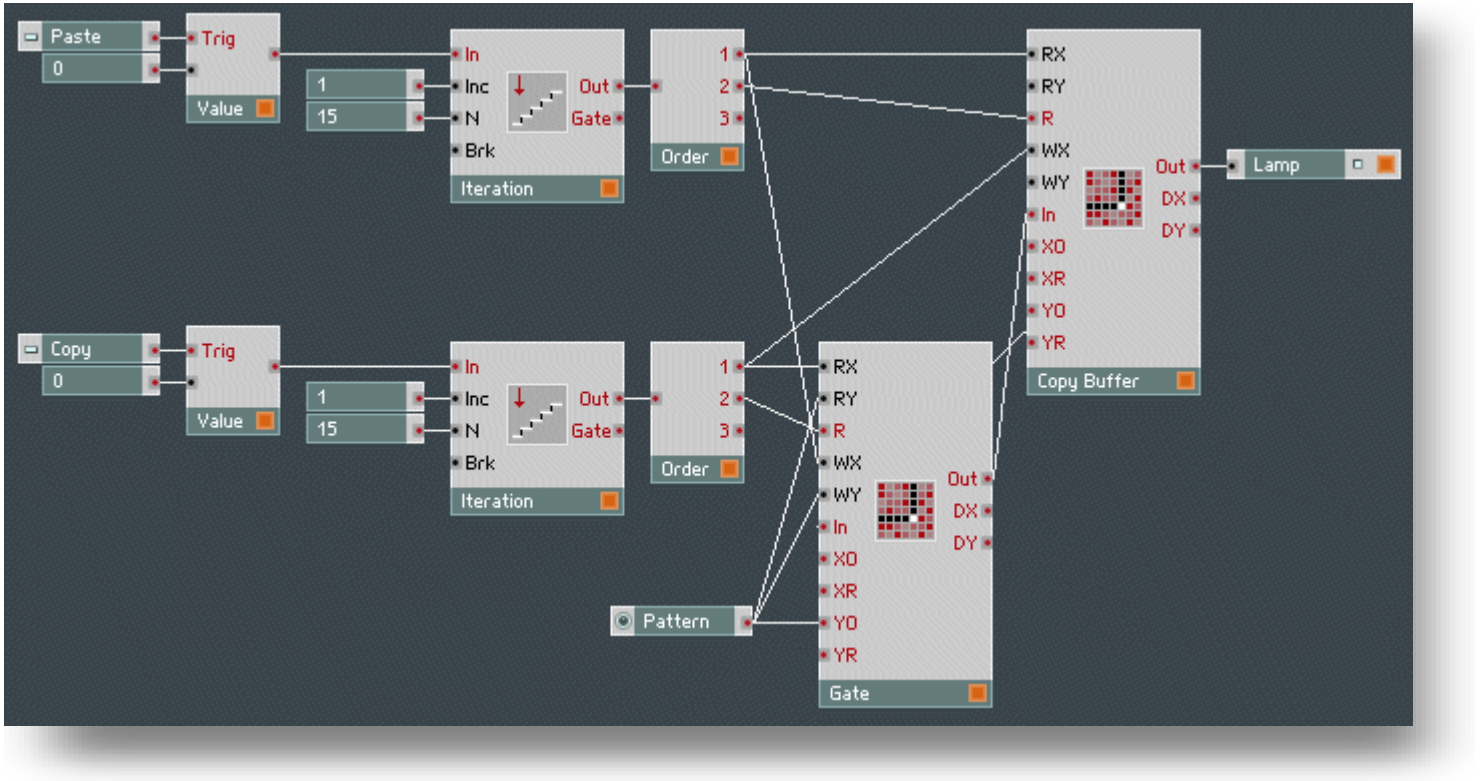

6
+
12

Feature interactions

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

Forcing demand in Reaktor



Feature interactions

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

User problems with dynamic references

From an Excel forum:

"No #REF! error, the cell just doesn't update with the new value (just stays exactly the same), even though the reference is correct and the referenced cell is obviously updated."

"I actually tried rebooting, didn't help."

From the LabVIEW docs:

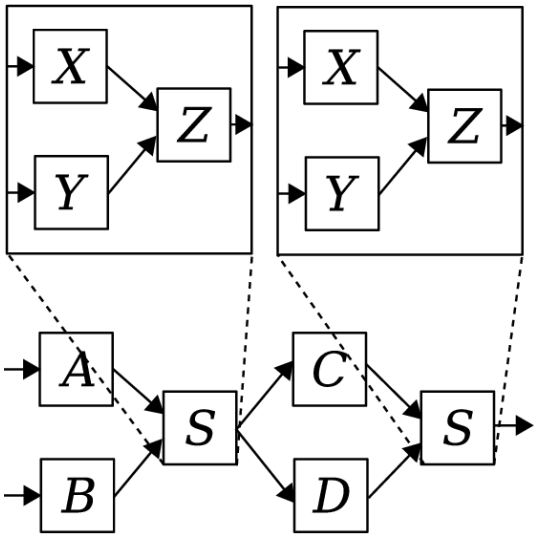
"This is a documented known issue that occurs in LabVIEW Real-Time versions 2014 and 2015. After making a modification to the VI, Error 1055 is thrown from any property node attempting to access the dynamic refnum. In order to resolve this error, close and re-open the VI."

Subprogram abstraction

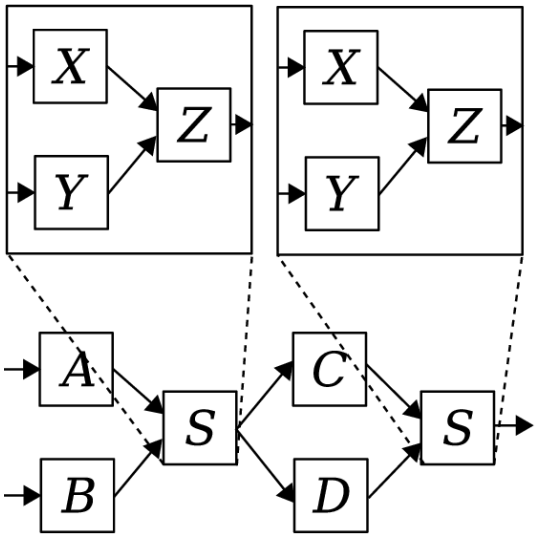
Design alternatives [Hil92]	**Pure Data**	**Excel**	**LabVIEW**	**Reaktor**	**VEE**	**Blender**
Box-line representation	Yes	No	Yes	Yes	Yes	Yes
Iteration	Yes (cycles)	Limited	Yes (construct)	Limited	Yes	No
Subprogram abstraction	Yes	No	Yes	Yes	Yes	Yes
Selector/distributor	Yes	Yes	Yes	Yes	Yes	Yes
Flow 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

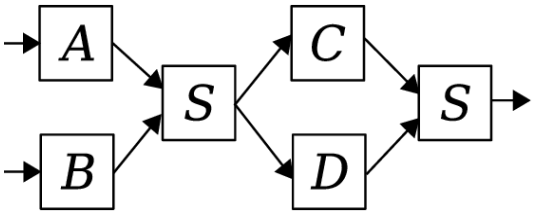
Subprograms



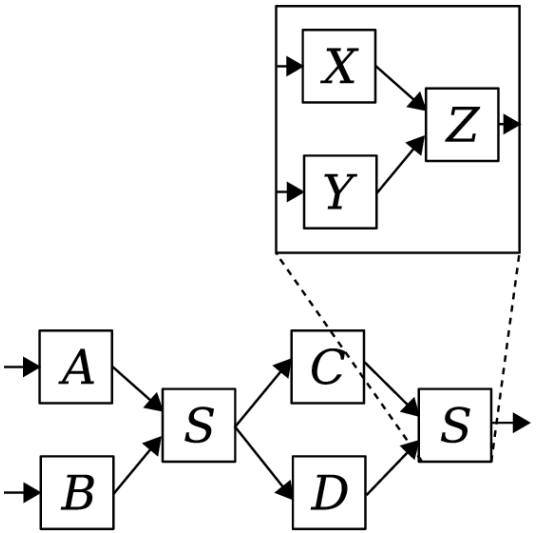
Subprograms



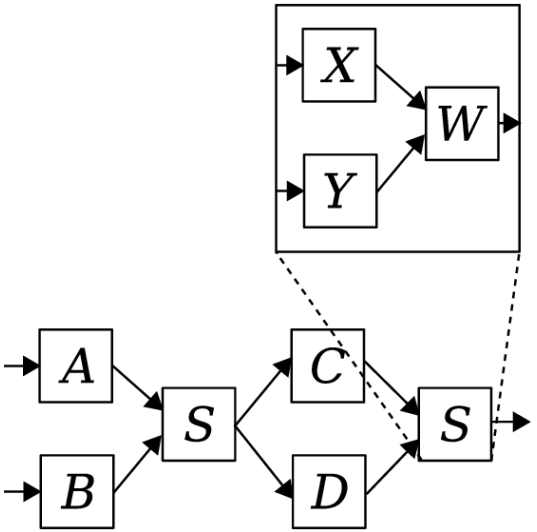
Subprograms



Subprograms

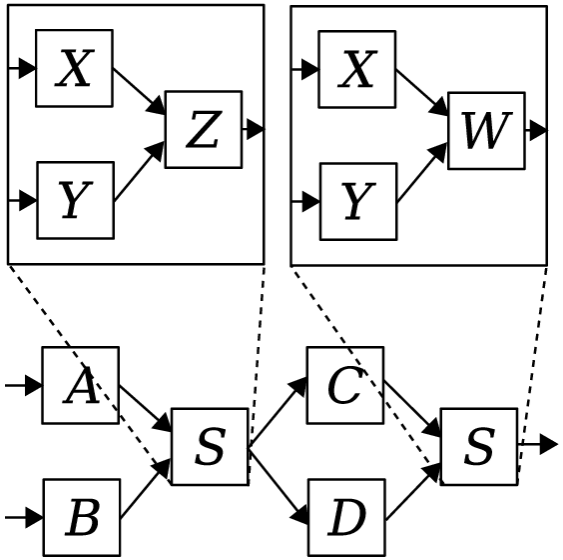


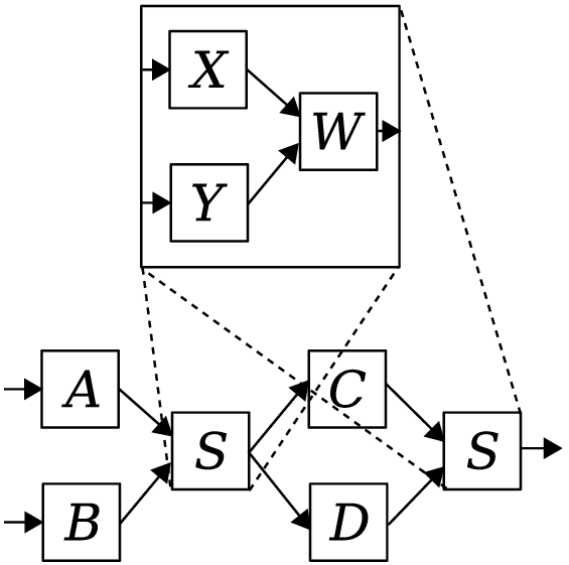
Subprograms



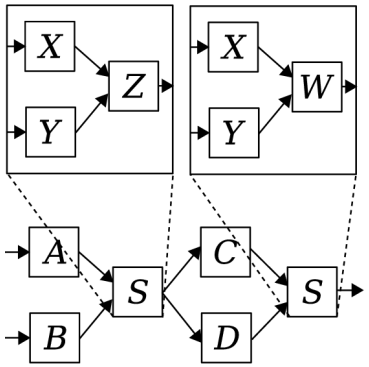
After editing

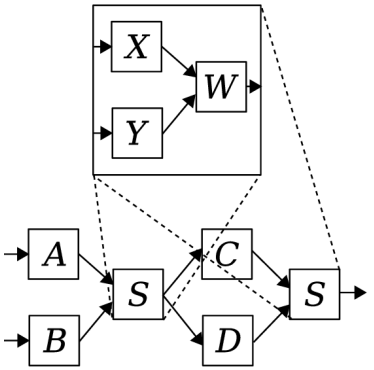
Two options:



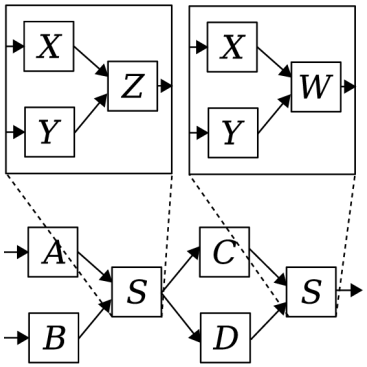


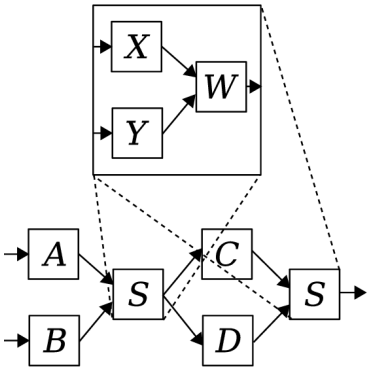
Runtime behavior

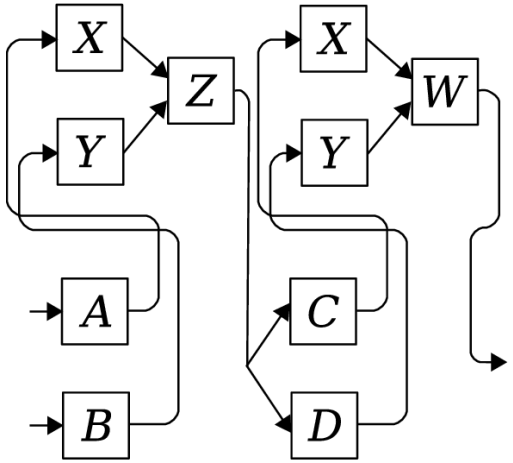




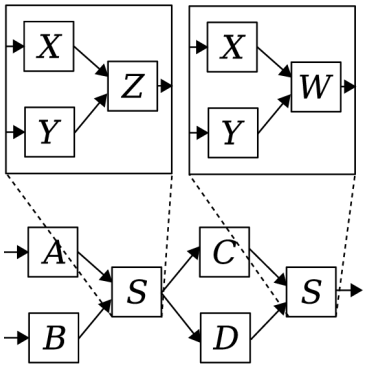
Runtime behavior

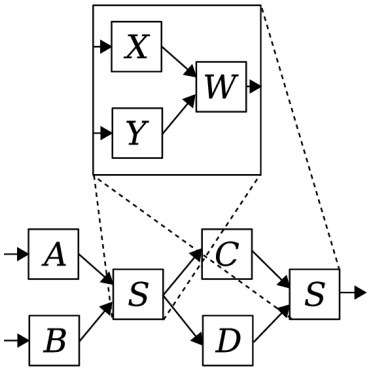


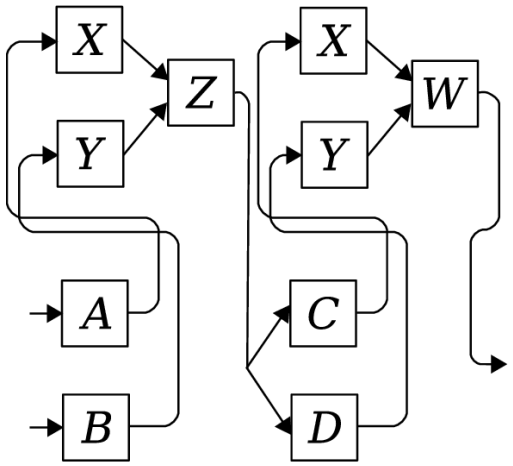


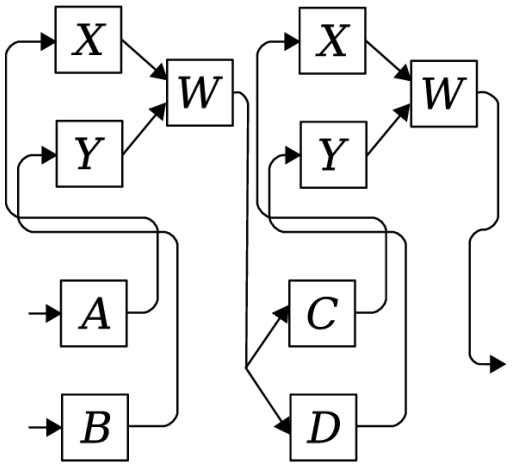


Runtime behavior

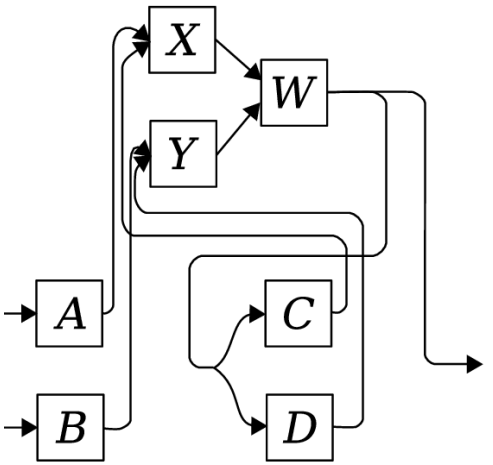








Reentrant...



...or not

Liveness

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

Liveness

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

Conclusions

Scripting vs. UI-Level languages

- UI-Level languages are today where scripting languages were in the 1980s

Scripting vs. UI-Level languages

- UI-Level languages are today where scripting languages were in the 1980s
- We aimed to get a glimpse of possible paths for a similar evolution
  - Reusable languages? Building blocks?

Scripting vs. UI-Level languages

- UI-Level languages are today where scripting languages were in the 1980s
- We aimed to get a glimpse of possible paths for a similar evolution
  - Reusable languages? Building blocks?
- We needed to step back to get a better understanding of these languages

Contributions

Mapping the design space of dataflow end-user language semantics

Lifting the veil on their underlying complexity

Contributions

Mapping the design space of dataflow end-user language semantics

Lifting the veil on their underlying complexity

A critique of design alternatives for dataflow end-user languages

A conceptual framework for comparing languages

Contributions

Mapping the design space of dataflow end-user language semantics

Lifting the veil on their underlying complexity

A critique of design alternatives for dataflow end-user languages

A conceptual framework for comparing languages

Identifying interdependencies in dataflow design choices

A sound design cannot be achieved combining building blocks at will

Secondary contributions

- A specification of realistic spreadsheet semantics
- Executable models of Pure Data and LabVIEW
- Insights on multi-language application architecture

In short

- End-user programming is more than scripting:
  UI-level programmability
  is often overlooked but it reaches all users

In short

- End-user programming is more than scripting:
  UI-level programmability
  is often overlooked but it reaches all users

- …but it needs to be approached as
  programming language design

In short

- End-user programming is more than scripting:
  UI-level programmability
  is often overlooked but it reaches all users

- ...but it needs to be approached as
  programming language design

- *Dataflow* is a proven approach for
  programmable UIs, but there is
  much room for evolution

Thank you!

http://hisham.hm/thesis