A Rationale for Faust Design Decisions

Yann Orlarey¹, Stéphane Letz¹, Dominique Fober¹, Albert Gräf², Pierre Jouvelot³

GRAME, Centre National de Création Musicale, France¹, Computer Music Research Group, U. of Mainz, Germany², MINES ParisTech, PSL Research University, France³

DSLDI, October 20, 2014
1-Music DSLs Quick Overview
### Some Music DSLs

- 4CED
- Adagio
- AML
- AMPLE
- Arctic
- Autoklang
- Bang
- Canon
- CHANT
- Chuck
- CLCE
- CMIX
- Cmusic
- CMUSIC
- Common Lisp Music
- Common Music
- Common Music Notation
- Csound
- CyberBand
- DARMS
- DCMP
- DMIX
- Elody
- EsAC
- Euterpea
- Extempore
- Faust
- Flavors Band
- Fluxus
- FOIL
- FORMES
- FORMULA
- Fugue
- Gibber
- GROOVE
- GUIDO
- HARP
- Haskore
- HMSL
- INV
- invokator
- KERN
- Keynote
- LPC
- Mars
- MASC
- Max
- MidiLisp
- MidiLogo
- MODE
- MOM
- Moxc
- MSX
- MUS10
- MUS8
- MUSCMP
- MuseData
- MusES
- MUSIC 10
- MUSIC 11
- MUSIC 360
- MUSIC 4B
- MUSIC 4BF
- MUSIC 4F
- MUSIC 6
- MCL
- MUSIC III/IV/V
- MusicLogo
- Music1000
- MUSIC7
- Musicetex
- MUSIGOL
- MusicXML
- Musixtex
- NIFF
- NOTELIST
- Nyquist
- OPAL
- OpenMusic
- Organum1
- Outperform
- Overtone
- PE
- Patchwork
- PILE
- Pla
- PLACOMP
- PLAY1
- PLAY2
- PMX
- POCO
- POD6
- POD7
- PROD
- Puredata
- PWGL
- Ravel
- SALIERI
- SCORE
- ScoreFile
- SCRIPT
- SIREN
- SMDL
- SMOKE
- SSP
- SSSP
- ST
- Supercollider
- Symbolic Composer
First Music DSLs, Music III/IV/V (Max Mathews)

- 1960: Music III introduces the concept of Unit Generators
- 1963: Music IV, a port of Music III using a macro assembler
- 1968: Music V written in Fortran (inner loops of UG in assembler)

```
ins 0 FM;
osc bl p9 p10 f2 d;
adn bl bl p8;
osc bl bl p7 f1 d;
adn bl bl p6;
osc b2 p5 p10 f3 d;
osc bl b2 bl f1 d;
out bl;
```

*FM synthesis coded in CMusic*
Csound

Originally developed by Barry Vercoe in 1985, Csound is today "a sound design, music synthesis and signal processing system, providing facilities for composition and performance over a wide range of platforms." (see http://www.csounds.com)

Example of Csound instrument

```
instr 2

  a1 oscil p4, p5, 1 ; p4=amp
  out a1 ; p5=freq
endin
```

Example of Csound score

```
f1 0 4096 10 1 ; sine wave

; ins strt dur amp(p4) freq(p5)
i2 0 1 2000 880
i2 1.5 1 4000 440
i2 3 1 8000 220
i2 4.5 1 16000 110
i2 6 1 32000 55
```

Max (Miller Puckette, 1987) is visual programming language for real-time audio synthesis and algorithmic composition with multimedia capabilities. It is named Max in honor of Max Mathews. It was initially developed at IRCAM. Since 1999 Max has been developed and commercialized by Cycling74. (see http://cycling74.com/)
SuperCollider

SuperCollider (James McCartney, 1996) is an open source environment and programming language for real time audio synthesis and algorithmic composition. It provides an interpreted object-oriented language that functions as a network client to a state of the art, real-time sound synthesis server. (see http://supercollider.sourceforge.net/)
Pure Data (Miller Puckette, 1996) is an open source visual programming language of the Max family. "Pd enables musicians, visual artists, performers, researchers, and developers to create software graphically, without writing lines of code". (see http://puredata.info/)
Elody

Elody (Fober, Letz, Orlarey, 1997) is a music composition environment developed in Java. The heart of Elody is a music language that embeds the lambda-calculus. The language expressions are handled through visual constructors and Drag and Drop actions allowing the user to play in real-time with the language.
OpenMusic

OpenMusic (Agon et al. 1998) is a music composition environment embedded in Common Lisp. It introduces a powerful visual syntax to Lisp and provides composers with a large number of composition tools and libraries.
ChucK (Ge Wang, Perry Cook 2003) is a concurrent, on-the-fly, audio programming language. It offers a powerful and flexible programming tool for building and experimenting with complex audio synthesis programs, and real-time interactive control. (see http://chuck.cs.princeton.edu)

// make our patch
SinOsc s => dac;

// time-loop, in which the osc’s frequency
// is changed every 100 ms
while( true ) {
    100::ms => now;
    Std.rand2f(30.0, 1000.0) => s.freq;
}
Live Coding

Live Coding is programming live, on stage, as an artistic performance.
Reactable

The Reactable is a tangible programmable synthesizer. It was conceived in 2003 by Sergi Jordà, Martin Kaltenbrunner, Günter Geiger and Marcos Alonso at the Pompeu Fabra University in Barcelona.
2-Faust Overview
Faust (Orlarey, Letz, Fober 2002) is a Domain-Specific Language for real-time signal processing and synthesis (like Csound, Max/MSP, Supercollider, Puredata, ...).

A Faust program denotes a signal processor: a (continuous) function that maps input signals to output signals.

Programming in Faust is essentially combining signal processors using an algebra of 5 composition operations:

```
process = noise*hslider("level",0,0,1,0.01);
nnoise = +(12345)~*(1103515245):(2147483647.0);
```
Brief overview to Faust

http://faust.gramex.fr

- Faust offers end-users a high-level alternative to C to develop audio applications for a large variety of platforms, from desktop to web applications, from audio plug-ins to embedded systems.

- The role of the Faust compiler is to synthesize the most efficient implementations for the target language (C, C++, LLVM, Javascript, etc.).

- Faust is used on stage for concerts and artistic productions, for education and research, for open sources projects and commercial applications:
3-Composing Signal Processors
Faust programs are *signal processors*

- A Faust program denotes a *signal processor* $p : \mathbb{S}^n \rightarrow \mathbb{S}^m$, a (continuous) function that maps a group of $n$ input *signals* to a group of $m$ output *signals*.

- Two kinds of signals:
  - Integer signals: $\mathbb{S}_\mathbb{Z} = \mathbb{Z} \rightarrow \mathbb{Z}$
  - Floating-point signals: $\mathbb{S}_\mathbb{R} = \mathbb{Z} \rightarrow \mathbb{R}$
  - $\mathbb{S} = \mathbb{S}^\mathbb{Z} \cup \mathbb{S}^\mathbb{R}$

- The value of a Faust signal is always 0 before time 0:
  - $\forall s \in \mathbb{S}, s(t < 0) = 0$

- Programming in Faust is essentially composing signal processors together using an algebra of five composition operations: $<, :, >, :, \sim$
Some Primitive Signal Processors

- !: $S^1 \rightarrow S^0$
  $\lambda\langle x \rangle.\langle \rangle$ (cut)

- -: $S^1 \rightarrow S^1$
  $\lambda\langle x \rangle.\langle x \rangle$ (wire)

- 3: $S^0 \rightarrow S^1$
  $\lambda\langle \rangle.\langle \lambda t. \begin{cases} 0 & t < 0 \\ 3 & t \geq 0 \end{cases} \rangle$ (number)

- +: $S^2 \rightarrow S^1$
  $\lambda\langle x, y \rangle.\langle \lambda t. x(t) + y(t) \rangle$ (addition)

- @: $S^2 \rightarrow S^1$
  $\lambda\langle x, y \rangle.\langle \lambda t. x(t - y(t)) \rangle$ (delay)
Composition Operations

- $(A, B)$ parallel composition
- $(A:B)$ sequential composition
- $(A<:B)$ split composition
- $(A:>B)$ merge composition
- $(A~B)$ recursive composition
Composition Operations

Parallel Composition

The *parallel composition* \((A, B)\) is probably the simplest one. It places the two block-diagrams one on top of the other, without connections.

\[(A, B): (S^n \rightarrow S^m) \rightarrow (S^{n'} \rightarrow S^{m'}) \rightarrow (S^{n+n'} \rightarrow S^{m+m'})\]

**Figure 1:** Example of parallel composition \((10, \ast)\)
Composition Operations
Sequential Composition

The *sequential composition* \((A:B)\) connects the outputs of \(A\) to the corresponding inputs of \(B\).

\[(A:B): (S^n \rightarrow S^m) \rightarrow (S^m \rightarrow S^p) \rightarrow (S^n \rightarrow S^p)\]

**Figure 2**: Example of sequential composition \(((\ast,/) : +)\)
Composition Operations

Split Composition

The *split composition* \((A<:B)\) operator is used to distribute the outputs of \(A\) to the inputs of \(B\).

\[(A<:B): (S^n \rightarrow S^m) \rightarrow (S^{k.m} \rightarrow S^p) \rightarrow (S^n \rightarrow S^p)\]

**Figure 3** : Example of split composition \(((10,20) <: (+,*,/))\)
Composition Operations

Merge Composition

The *merge composition* \((A : > B)\) is used to connect several outputs of \(A\) to the same inputs of \(B\). Signals connected to the same input are added.

\[(A : > B): (S^n \rightarrow S^{k.m}) \rightarrow (S^m \rightarrow S^p) \rightarrow (S^n \rightarrow S^p)\]

**Figure 4:** Example of merge composition \(((10, 20, 30, 40) : > *)\)
The *recursive composition* \((A \sim B)\) is used to create cycles in the block-diagram in order to express recursive computations.

\[
(A \sim B) : (S^n + n' \rightarrow S^m + m') \rightarrow (S^m' \rightarrow S^n') \rightarrow (S^n \rightarrow S^m + m')
\]

**Figure 5**: Example of recursive composition \(+ (12345) \sim *(1103515245)\)
A Very Simple Example

\[ y(t) = \begin{cases} 
0 & t < 0 \\
1 + y(t - 1) = 1 + t & t \geq 0
\end{cases} \]
A Mixer Channel

Figure 6: Source code of a simple mixer channel

```cpp
// Simple 1-voice mixer with mute button, volume control
// and stereo pan

process = vgroup("voice", mute : amplify : pan);

mute = *(1-checkbox("[3]mute"));
amplify = *(vslider("[2]gain", 0, 0, 1, 0.01));
pan = *(1-p)
with {
  p = nentry("[1]pan[style:knob]", 0.5, 0, 1, 0.1);
}
```

Figure 7: Resulting application
4-Easy Deployment
Faust Architecture System
Easy deployment: one Faust code, multiple targets

- Chuck
- Max/MSP
- VST
- Alsa
- Coreaudio
- Ladspa
- libsndfile
- PortAudio
- ActionScript
- Pure Data
- SuperCollider
- Pure
- Matlab
- CSound
- Q
- etc...
Faust Architecture System

Separation of concern

The *architecture file* describes how to connect the audio computation to the external world.
Faust Architecture System
Examples of supported architectures

- Audio plugins:
  - AudioUnit
  - LADSPA
  - DSSI
  - LV2
  - Max/MSP
  - VST
  - Pure Data
  - Csound
  - SuperCollider
  - Pure
  - Chuck
  - Octave
  - Flash

- Audio drivers:
  - Jack
  - Alsa
  - CoreAudio
  - Web Audio API

- Graphic User Interfaces:
  - QT
  - GTK
  - Android
  - iOS
  - HTML5/SVG

- Other User Interfaces:
  - OSC
  - HTTPD
5-Compiler/Code Generation
Faust Compiler
Main phases of the compiler

- Faust Program
- evaluation
- Block-Diagram in Normal Form
- symbolic propagation
- Signal Equations
- normalization
- Signal Equations in Normal Form
- type inference
- Typed Signals
- code generation
- Implementation Code (C++)
Faust Compiler
Four code generation modes

- **Parallel code generator** (OpenMP directives)
- **Parallel code generator** (Work Stealing Scheduler)

- **Vector code generator** (loop separation)

- **Scalar code generator**
6-Performance
Performance of the Generated Code
How the C++ code generated by FAUST compares with hand written C++ code.

STK vs FAUST (CPU load)

<table>
<thead>
<tr>
<th>File name</th>
<th>STK</th>
<th>FAUST</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>blowBottle.dsp</td>
<td>3.23</td>
<td>2.49</td>
<td>-22%</td>
</tr>
<tr>
<td>blowHole.dsp</td>
<td>2.70</td>
<td>1.75</td>
<td>-35%</td>
</tr>
<tr>
<td>bowed.dsp</td>
<td>2.78</td>
<td>2.28</td>
<td>-17%</td>
</tr>
<tr>
<td>brass.dsp</td>
<td>10.15</td>
<td>2.01</td>
<td>-80%</td>
</tr>
<tr>
<td>clarinet.dsp</td>
<td>2.26</td>
<td>1.19</td>
<td>-47%</td>
</tr>
<tr>
<td>flustestk.dsp</td>
<td>2.16</td>
<td>1.13</td>
<td>-47%</td>
</tr>
<tr>
<td>saxophony.dsp</td>
<td>2.38</td>
<td>1.47</td>
<td>-38%</td>
</tr>
<tr>
<td>sitar.dsp</td>
<td>1.59</td>
<td>1.11</td>
<td>-30%</td>
</tr>
<tr>
<td>tibetanBowl.dsp</td>
<td>5.74</td>
<td>2.87</td>
<td>-50%</td>
</tr>
</tbody>
</table>

Overall improvement of about 41% in favor of FAUST.
Performance of the Generated Code

What improvements to expect from parallelized code?

Sonik Cube

Compared performances of the various C++ code generation strategies according to the number of cores:
7-Tools
Tools

Faust ecosystem

**local tools**

- `faust` command line compiler (lin, osx)
- `faust2puredata`
- `faust2max6`
- `faust2vst`
- `faust2android`
- `faust2ios`
- command line builders

**FaustWorks**
IDE (lin, osx)

**faustgen~**
eMBEDDED Faust compiler for max (osx, win)

**faustcompile**
eMBEDDED Faust compiler for csound6 (lin, osx, win)

**web servers**

- `http://faust.grame.fr/index.php/online-examples`
  Online IDE (any browser)

- `http://faustservice.grame.fr/`
  Compiler API (used by FaustLive)
Tools
Available compilers

- **Command line tools**
  - `faust` command line
  - `faust2xxx` command line

- **Web based tools**
  - Online Compiler (http://faust.grame.fr)
  - Faustweb API (http://faustservice.grame.fr)

- **Embedded compiler (libfaust)**
  - Faustgen for Max/MSP
  - Faustcompile, etc. for Csound (V. Lazzarini)
  - Faustnode for the Web Audio API
  - Antescofo (IRCAM’s score follower)
  - LibAudioStream (Audio renderer)
  - iScore (LaBRI)

- **IDE**
  - FaustWorks (requires Faust)
  - FaustLive (self contained)
Some useful libraries

- math.lib
- music.lib, imports math.lib
- hoa.lib, imports math.lib
- filter.lib, imports music.lib
- effect.lib, imports filter.lib
- oscillator.lib, imports filter.lib
Tools

Some useful links

- Website and online compiler:

- Faust distribution:
  - http://sourceforge.net/projects/faudiostream/
  - git clone
    - git://faudiostream.git.sourceforge.net/gitroot/faudiostream/faudiostream
  - cd faudiostream; make; sudo make install

- FaustWorks:
  - http://sourceforge.net/projects/faudiostream/
  - git clone git://faudiostream.git.sourceforge.net/gitroot/faudiostream/FaustWorks

- FaustLive:
  - http://sourceforge.net/projects/faudiostream/
  - git clone git://faudiostream.git.sourceforge.net/gitroot/faudiostream/faustlive
8-To Summarize
To Summarize

FAUST is a DSL for real-time signal processing and synthesis. Its design is based on several principles:

- High-level specification language
- End-Users oriented
- Simple well-defined formal semantics
- Purely synchronous functional approach
- Textual, "block-diagram oriented", syntax
- Favors reuse and composition of existing programs,
- Efficient sample level processing
- Fully compiled code (sequential or parallel)
- Embeddable code
- Easy deployment: single code multiple targets
- Preservable via automatic documentation
9-Bibliography
Bibliography

10-Acknowledgments
Acknowledgments

This research was funded in part by the Agence nationale pour la recherche (ANR) via the following projects:

- ASTREE [2008-CORD-003]
- INEDIT [ANR 2012 CORD 009]
- FEEVER [ANR-13-BS02-0008]

Many persons have been contributing to the FAUST project by providing code for the compiler, architecture files, libraries, examples, documentation, scripts, bug reports, ideas, etc. We would like to thank them and especially (in alphabetic order): Fons Adriaensen, Karim Barkati, Jerome Barthelemy, Tim Blechmann, Tiziano Bole, Alain Bonardi, Myles Borins, Baktery Chanka, Thomas Charbonnel, Raffaele Ciavarella, Julien Colafrancesco, Damien Cramet, Sarah Denoux, Robin Gareus, Etienne Gaudrin, Olivier Guillerminet, Albert Graef, Pierre Guillot, Olivier Guillerminet, Pierre Jouvelot, Stefan Kersten, Victor Lazzarini, Matthieu Leberre, Christophe Lebreton, Mathieu Leroi, Fernando Lopez-Lezcano, Kjetil Mattheussen, Hermann Meyer, Romain Michon, Remy Muller, Eliott Paris, Reza Payami, Laurent Pottier, Sampo Savolainen, Nicolas Scaringella, Anne Sedes, Priyanka Shekar, Stephen Sinclair, Travis Skare, Julius Smith, Mike Solomon, Michael Wilson