Ingen: A Meta-Modular Plugin Environment

David Robillard

School of Computer Science
Carleton University

April 10, 2015

"Ingen"

- Nobody? Nothing?
- That company from Jurassic Park?
- ...
- Instrument Generator, "engine"

What is Ingen?

- Ingen is a modular synth, effects processor, mixer, router, MIDI processor, cheese grater, chain degreaser...
- ...a "modular"
- Designed around LV2 plugins
- Usable in many different contexts

Features

- Polyphonic
- Recursive (graphs within graphs)
- Many data types (including events like MIDI)
- Strict client/server architecture
  - Real-time editable
  - Flexible deployment

Philosophy

- Do one thing and do it well
- Integrate with the surrounding environment
- Make use of existing facilities

Existing Facilities?

- Plugins!
- Internals Considered Harmful

Surrounding Environment

In the Lignux audio world, there are several choices:

- JACK application
- LV2 plugin
- Network service
- Physical device (e.g. MOD)

Philausophy

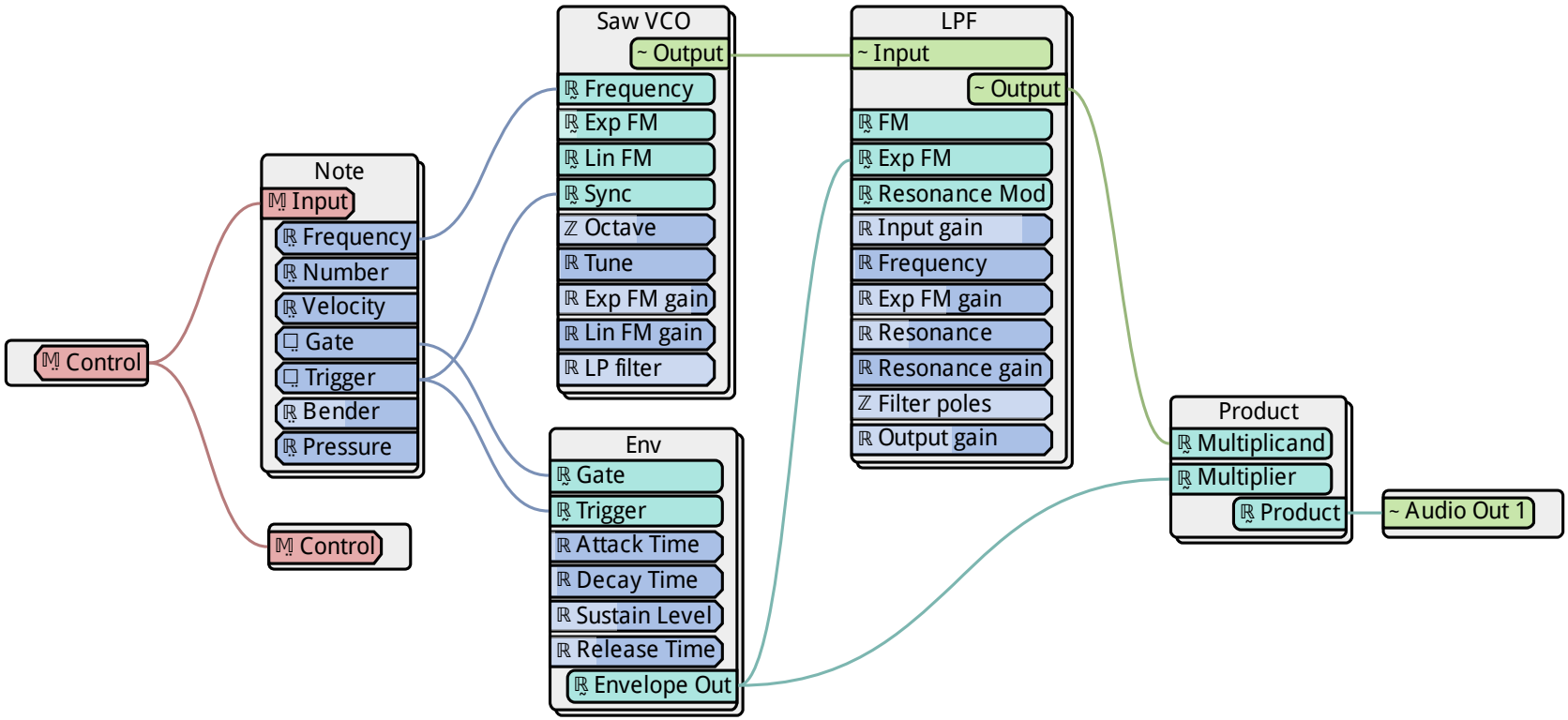
- Do one thing and do it well and do it with real-time safety
- Keep GUIs at arm's length

Terminology

Ingen contains:

- Graphs
- Blocks
- Ports
- Arcs

A Graph


Saw VCO
~ Output
ℝ Frequency
ℝ Exp FM
ℝ Lin FM
ℝ Sync
ℤ Octave
ℝ Tune
ℝ Exp FM gain
ℝ Lin FM gain
ℝ LP filter
LPF
~ Input
~ Output
ℝ FM
ℝ Exp FM
ℝ Resonance Mod
ℝ Input gain
ℝ Frequency
ℝ Exp FM gain
ℝ Resonance
ℝ Resonance gain
ℤ Filter poles
ℝ Output gain
Note
𝕄 Input
ℝ Frequency
ℝ Number
ℝ Velocity
□ Gate
□ Trigger
ℝ Bender
ℝ Pressure
𝕄 Control
𝕄 Control
Env
ℝ Gate
ℝ Trigger
ℝ Attack Time
ℝ Decay Time
ℝ Sustain Level
ℝ Release Time
ℝ Envelope Out
Product
ℝ Multiplicand
ℝ Multiplier
ℝ Product
~ Audio Out 1

Data Model

Everything is described in a simple data model:

- All objects have a path, e.g. `/amp/gain`
- Objects are a dictionary of properties (key:value)
- Inherently extensible: keys can be added without breakage

Data Model Example

A snippet of an Ingen graph:

```
</osc>
  a                ingen:Block ;
  lv2:prototype    <urn:someplugin> ;
  ingen:canvasX    42.0 ;
  ingen:canvasY    24.0 .
```

Protocol

Ingen is controlled by manipulating the model:

- Changes are based on property manipulation
- Everything done with a few generic methods: get, set, put, ...
  - Vaguely HTTP-like
- No special "commands" for specific objects
  - No: `block.move_to`(42, 0)
  - Instead: `set` block.x = 42, block.y = 0

Simple Message Example

Moving a block on the canvas:

```
[
  a                patch:Set ;
  patch:subject    </osc> ;
  patch:property   ingen:canvasX ;
  patch:value      42.0 ;
]
```

Slightly Less Simple Message Example

To add, or put, a new block:

```
[
  a               patch:Put ;
  patch:subject </osc> ;
  patch:body [
    a               ingen:Block ;
    lv2:prototype <urn:someplugin> ;
    ingen:canvasX 42.0 ;
    ingen:canvasY 24.0 ;
  ]
]
```

Who Cares?

- Elegant correspondence between protocol and data model
- Identical syntax used on the wire and in saved files
- Conceptually simple

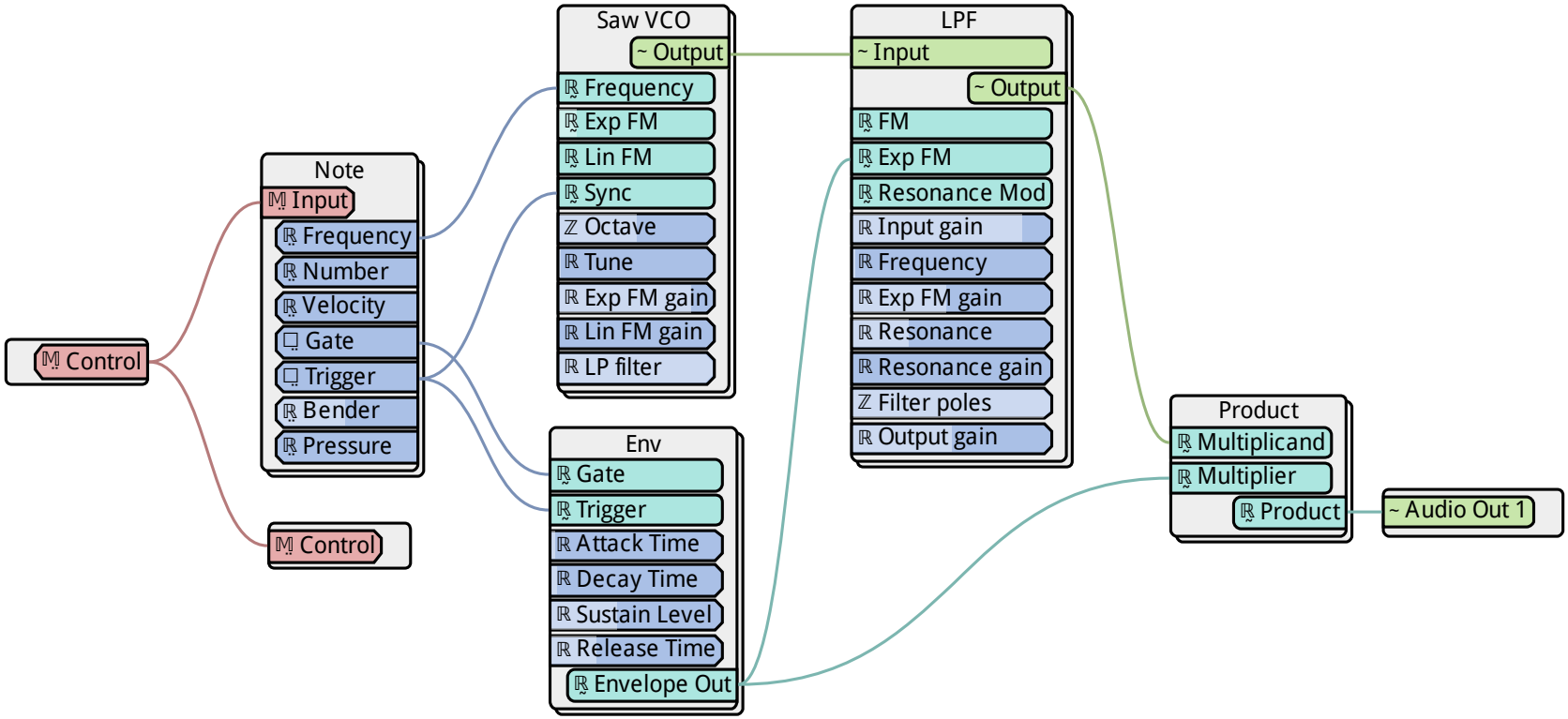
No, really... who other than you cares?

- Allows flexible deployment:
- Ingen as server, network controlled from a different machine
  - Dump text protocol for a plain text log of activity
- Run Ingen as an LV2 plugin, control via LV2 ports
  - Protocol in binary (LV2 atoms, native `float`, etc)
  - No shady underhanded communication
  - GUI talks to engine via ports like any LV2 plugin
- Extensible, properties can be added freely
- A few methods are capable of everything

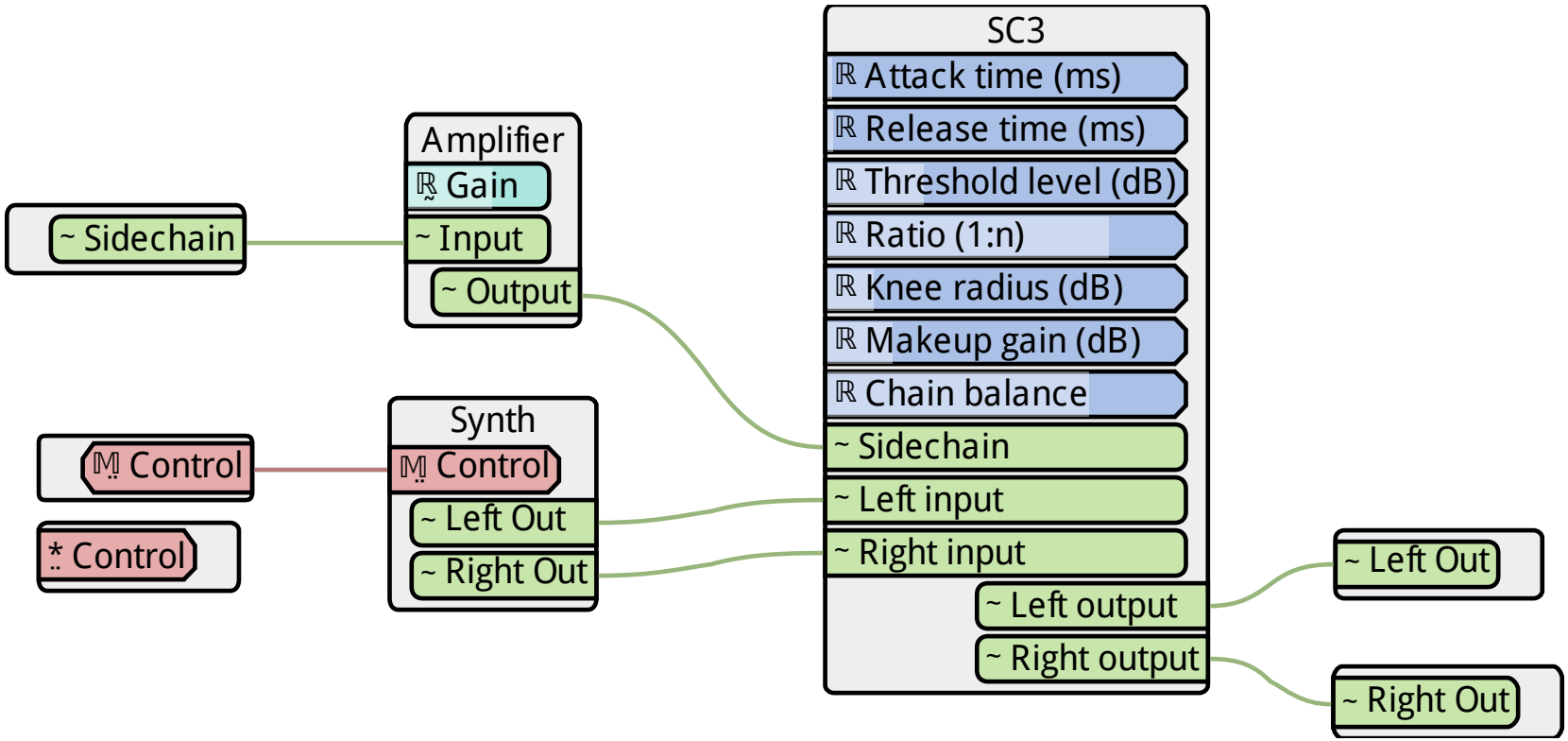
Scenarios

- Need to put some non-linear in another host (e.g. Ardour mixer strip)
- Remote-controllable "glue" for a headless JACK box
- Want to publish a plugin but can't or don't want to code
  - Ingen graphs are LV2 plugins
  - . . . literally. No special export, no compilation; the one and only save format is LV2 compatible
  - Save to `LV2_PATH` (typically `~/.lv2`) and graphs will be visible in any LV2 host

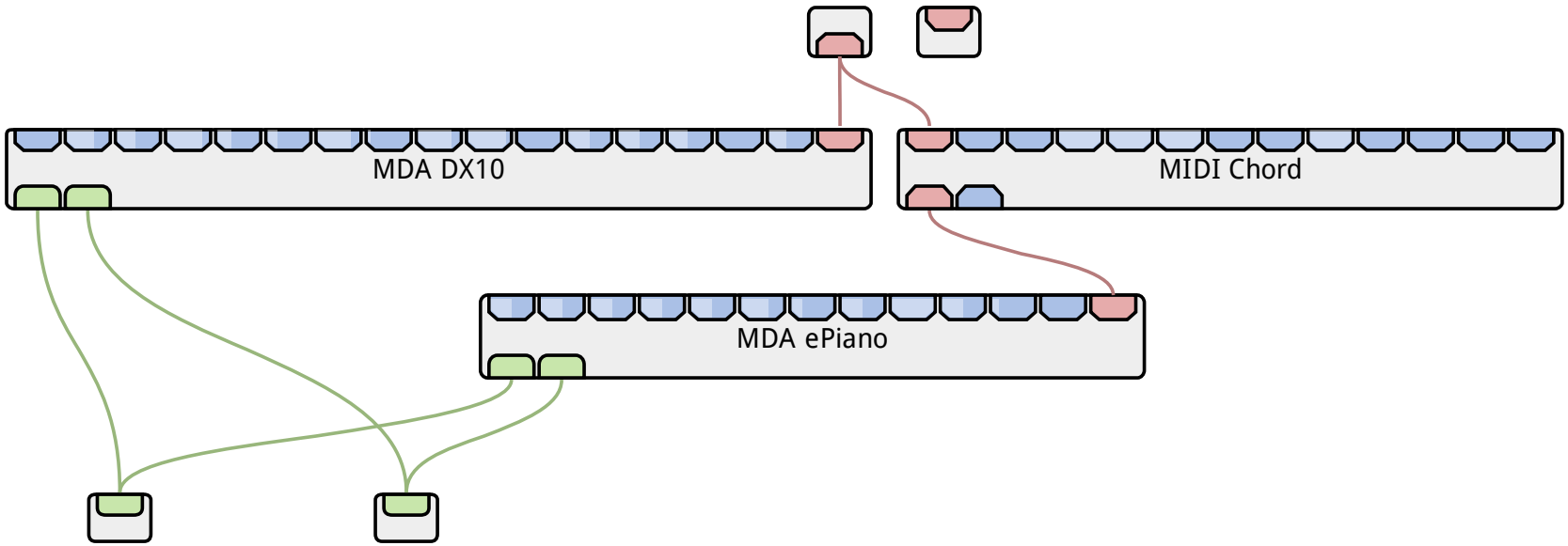
A Synth


Saw VCO
~ Output
Frequency
Exp FM
Lin FM
Sync
Octave
Tune
Exp FM gain
Lin FM gain
LP filter
LPF
~ Input
~ Output
FM
Exp FM
Resonance Mod
Input gain
Frequency
Exp FM gain
Resonance
Resonance gain
Filter poles
Output gain
Note
Input
Frequency
Number
Velocity
Gate
Trigger
Bender
Pressure
Control
Control
Env
Gate
Trigger
Attack Time
Decay Time
Sustain Level
Release Time
Envelope Out
Product
Multiplicand
Multiplier
Product
~ Audio Out 1

Flexible I/O (Sidechains)



MIDI Filtering



Mundane Improvements

- Performance / memory consumption improvements
- Better state/preset support
- Stability, more rigorous testing
- GUI polish

Whiz-Bang

- Control panel building (presentation mode)
- Novel presentation of patching canvas? (Pies?)
- Integrated online patch repository (MOD?)
- Dynamic ports when running as LV2 plugin
- Design plugin suite for message-based programming
- Make it as simple as possible for users to publish "plugins"