Going deeper with CLAP

Alexandre Bique & Urs Heckmann

Overview

- About
- Adoption
- Maturity and Innovation
- Some real world Benchmarks (Threadpool)
- Example: Integrating the Unified Event Queue
- Example: How we did Parameter Modulation (u-he)
- Q & A

About CLAP

- CLAP stands for CLever Audio Plug-in
- A plain C interface to define a standard for DAW and Audio Plug-In
- Simple, Robust, Modern and a Permissive License (MIT)
- Version 1.0.0 launched 2 years ago (on Jun 3, 2022)
- https://cleveraudio.org
- git clone https://github.com/free-audio/clap

Adoption in 2024

- Bitwig Studio, Reaper, FL Studio, Studio One, … (15+)
- u-he, FabFilter, TAL, Plogue, … (70+)
- iPlug2, DPF, JUCE (announcement), …
- +500% usage in Bitwig Studio Nov 23 - Nov 24

Maturity & Innovation

- CLAP is complete and stable
  - Meets standard expectations
- New CLAP Extensions
  - Undo host integration
  - State Conversion (Plug-In A to Plug-In B)
  - ...

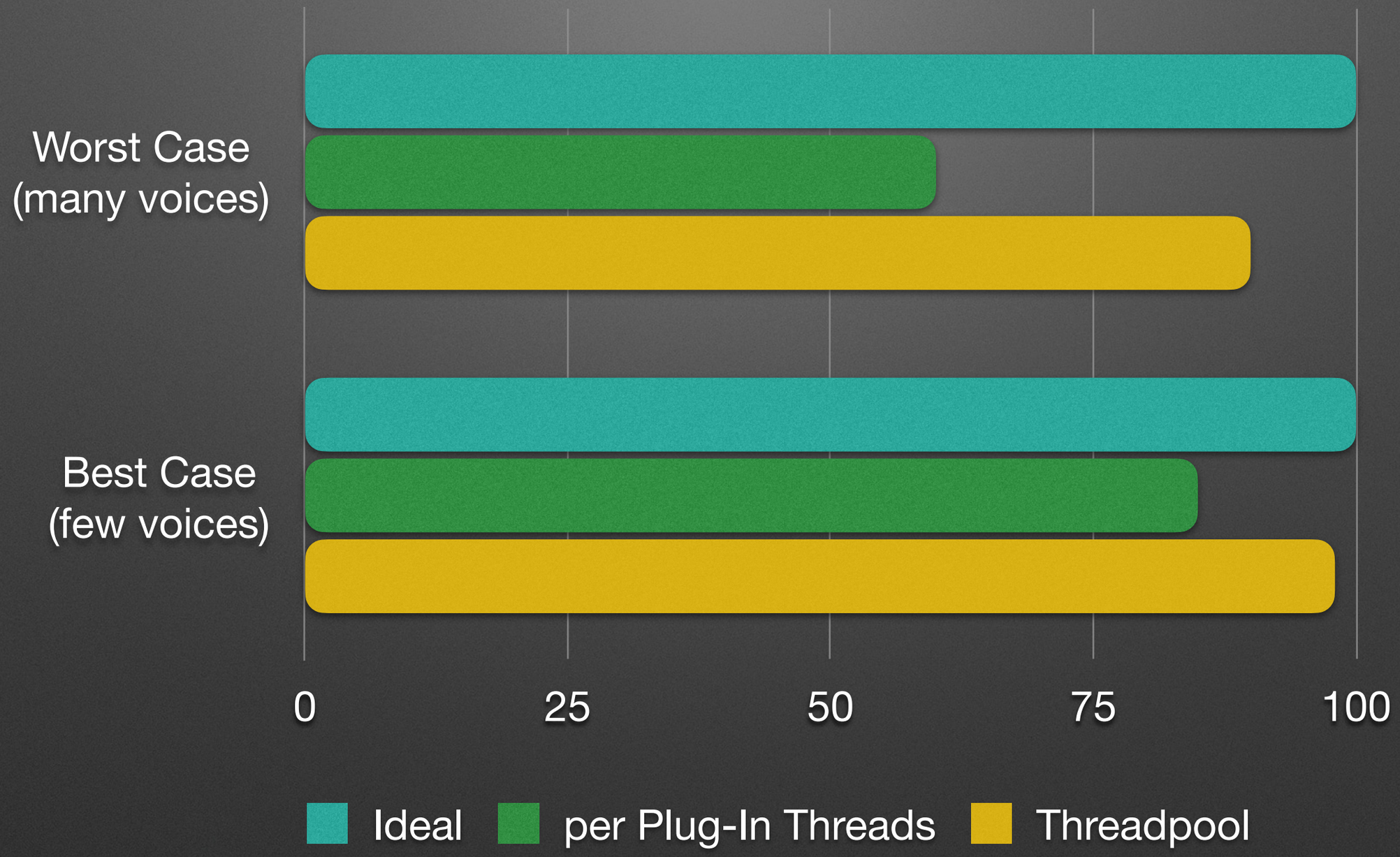
Threadpool

Make use of CPU Cores for performance hungry Plug-Ins

Threadpool

- Problem:
  - Some Plug-Ins benefit from Multithreaded Processing
  - Plug-In Threads compete with Host, other Plug-Ins
- Solution:
  - CLAP Threadpool extension allows Host to schedule Threads for Plug-Ins
  - Reduces Concurrence and Context Switches

Plug-In vs CLAP Threadpool



Method: Different multithreaded plug-ins, as many voices as possible.

A Unified Event Queue

Notes
Note Expressions
Parameters
Gestures
Transport
MIDI
...

clap / include / clap / **events.h**

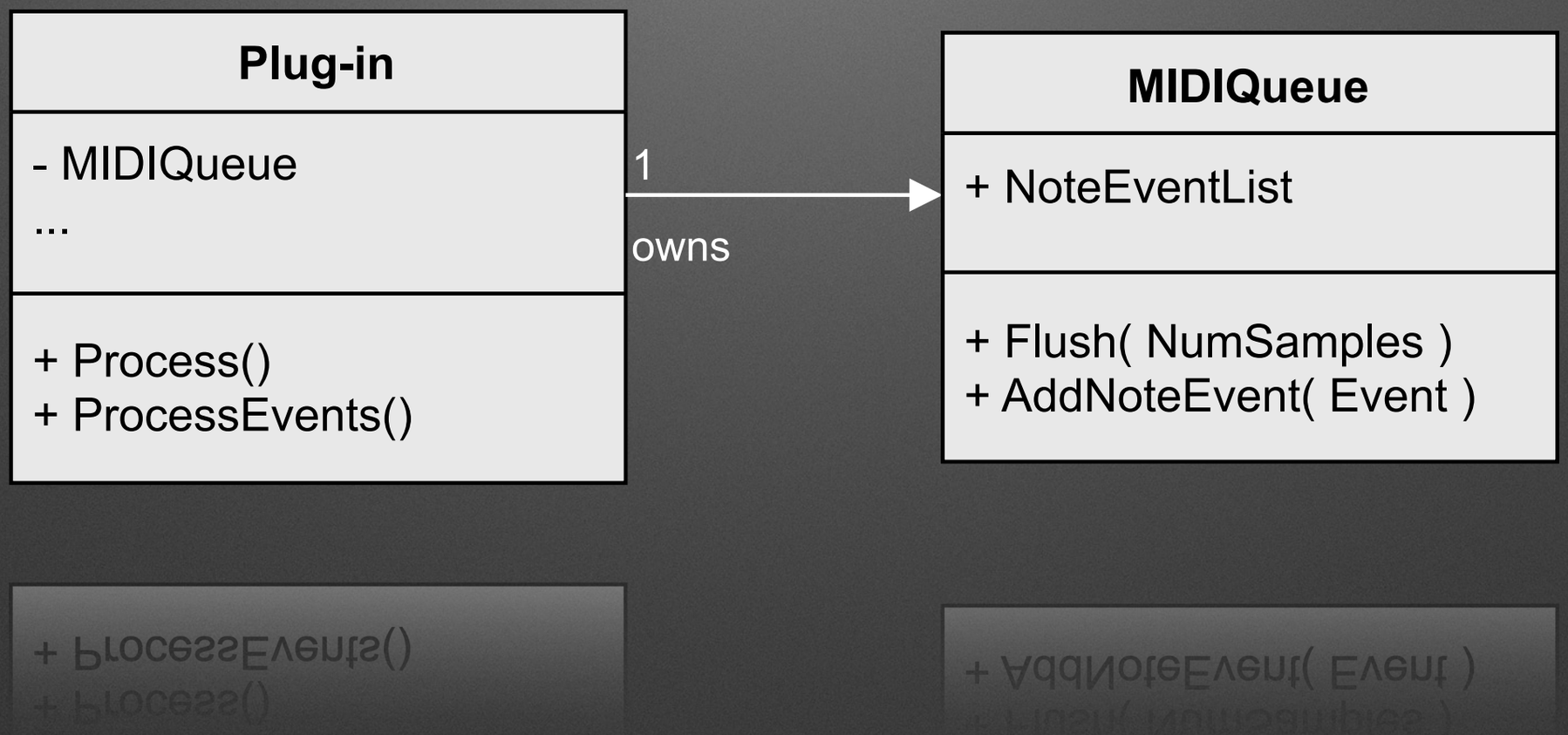
Code | Blame | 340 lines (289 loc) · 12.7 KB

```
85      // These four events use clap_event_note.
86      CLAP_EVENT_NOTE_ON = 0,
87      CLAP_EVENT_NOTE_OFF = 1,
88      CLAP_EVENT_NOTE_CHOKE = 2,
89      CLAP_EVENT_NOTE_END = 3,
90
91      // Represents a note expression.
92      // Uses clap_event_note_expression.
93      CLAP_EVENT_NOTE_EXPRESSION = 4,
94
95      // PARAM_VALUE sets the parameter's value; uses cla
96      // PARAM_MOD sets the parameter's modulation amount
97      //
98      // The value heard is: param_value + param_mod.
99      //
100     // In case of a concurrent global value/modulation
101     // the voice should only use the polyphonic one and
102     // amount will already include the monophonic signa
103     CLAP_EVENT_PARAM_VALUE = 5,
104     CLAP_EVENT_PARAM_MOD = 6,
105
106     // Indicates that the user started or finished adju
107     // This is not mandatory to wrap parameter changes
108     // the user experience a lot when recording automat
109     // Uses clap_event_param_gesture.
110     CLAP_EVENT_PARAM_GESTURE_BEGIN = 7,
111     CLAP_EVENT_PARAM_GESTURE_END = 8,
112
113     CLAP_EVENT_TRANSPORT = 9,   // update the transport
114     CLAP_EVENT_MIDI = 10,       // raw midi event; clap
115     CLAP_EVENT_MIDI_SYSEX = 11, // raw midi sysex event
116     CLAP_EVENT_MIDI2 = 12,      // raw midi 2 event; cl
117 };
```

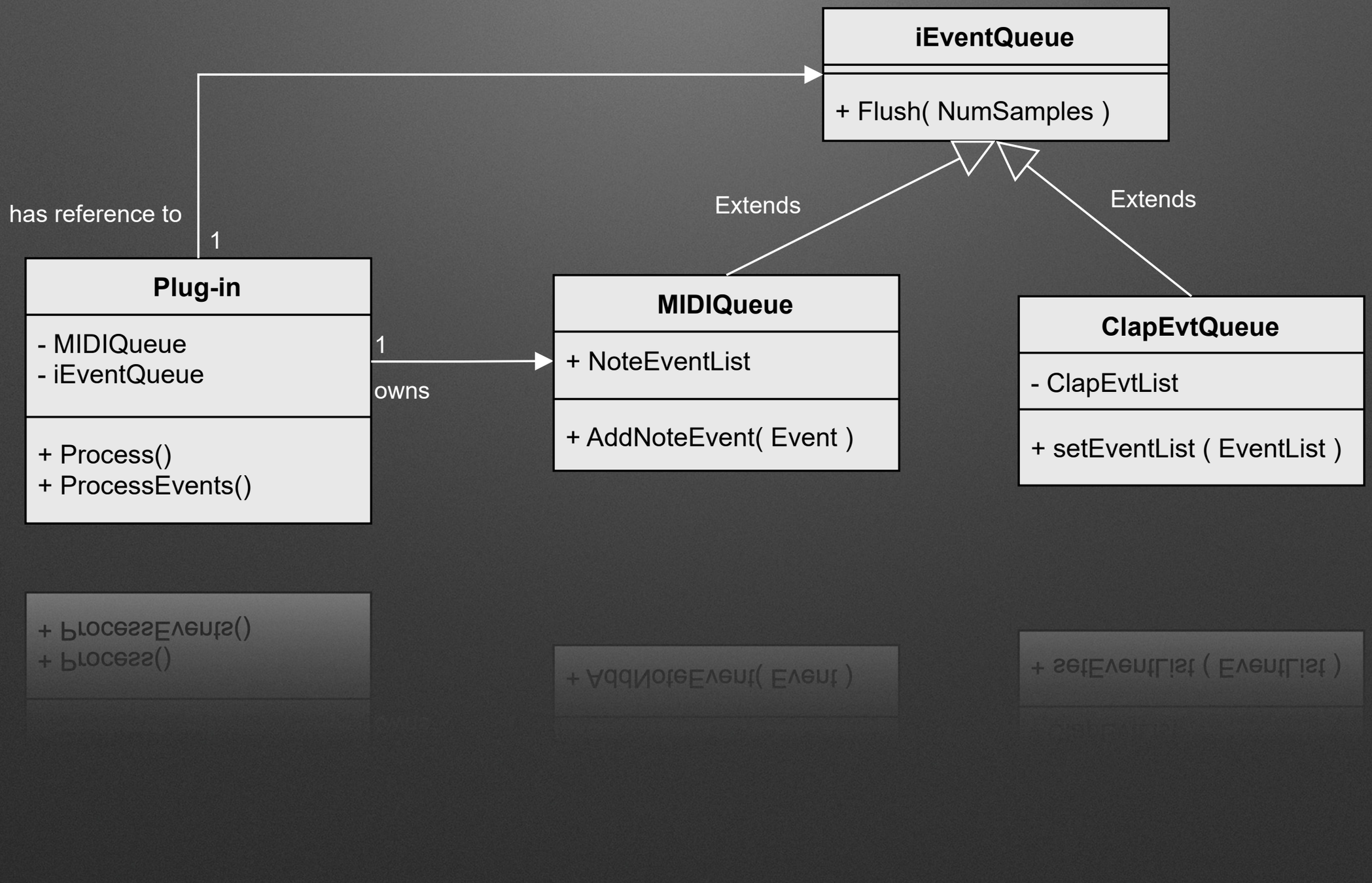
Unified Event Queue

- All Events in one Queue
- Parameter Automation, MIDI, Notes, Note Expressions, Parameter Modulation, Transport, Gestures, …
- Timestamped, Sorted, Non-Ambiguous
- Input and Output
- (Plug-Ins can manipulate Events and pass them on)

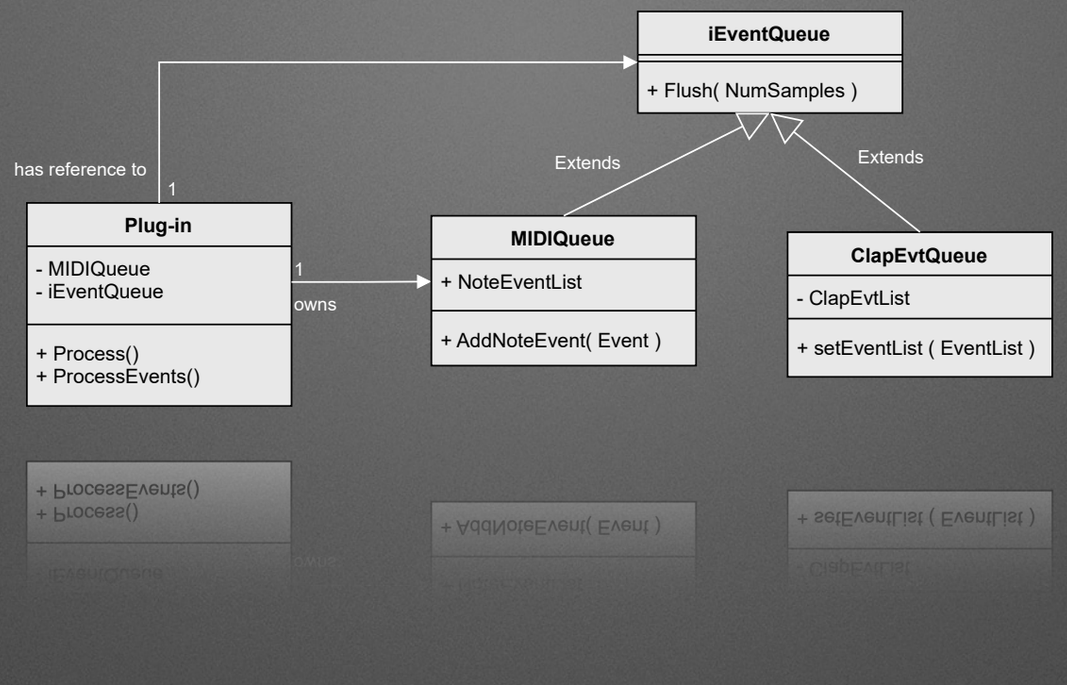
u-he
Before CLAP: MIDI Queue



Using CLAP Queue directly



Using CLAP Queue directly



- Uses CLAP Event List unchanged
- All Event Types evaluated at Control Rate
- In-Buffer Timing Changes
- Support for Parameter Modulation

Parameter Modulation

- Similar to Automation
- Temporary Offsets to Parameters
- Does not change Parameter Value in Plug-In State
- “Knob doesn’t turn”
- “go back to original settings”
- Can be polyphonic!

clap / include / clap / events.h

Code | Blame | 340 lines (289 loc) · 12.7 KB

```
234  typedef struct clap_event_param_mod {
235     clap_event_header_t header;
236
237     // target parameter
238     clap_id param_id; // @ref clap_param
239     void   *cookie;   // @ref clap_param
240
241     // target a specific note_id, port,
242     // -1 meaning wildcard, per the wild
243     int32_t note_id;
244     int16_t port_index;
245     int16_t channel;
246     int16_t key;
247
248     double amount; // modulation amount
249  } clap_event_param_mod_t;
250
```

Our Approach: Repurposing the ModMatrix

- ModMatrix has N Slots
- Each Slot takes a modulator (LFO, Velocity etc.) and routes it onto a Parameter
- Adjustable Modulation Depth
- Targets any Parameter that can be represented by a Knob or a Fader (float)
- Can “stack” Slots on Params

Context

+ Tempo
+ Samplerate
+ NumSamples
+ Buffers
...

owns 1

Plug-in

- Context
- RenderGraph
- Modules

+ Process()

1...n owns

owns 1

ModMatrixSlot

- TargetModule
- TargetParam
- ModSource
- ModDepth

+ PrepareModule(Module)
+ RestoreModule(Module)

RenderGraph

- ModuleList
- ModMatrixSlots

+ RunModule (Module, Context)
+ CalculateProcessPath()

1...n triggers

Module (LFO, Filter)

- Parameter A
- Parameter B
...
- State

+ Run(Context)

owns 1...n

pseudo code

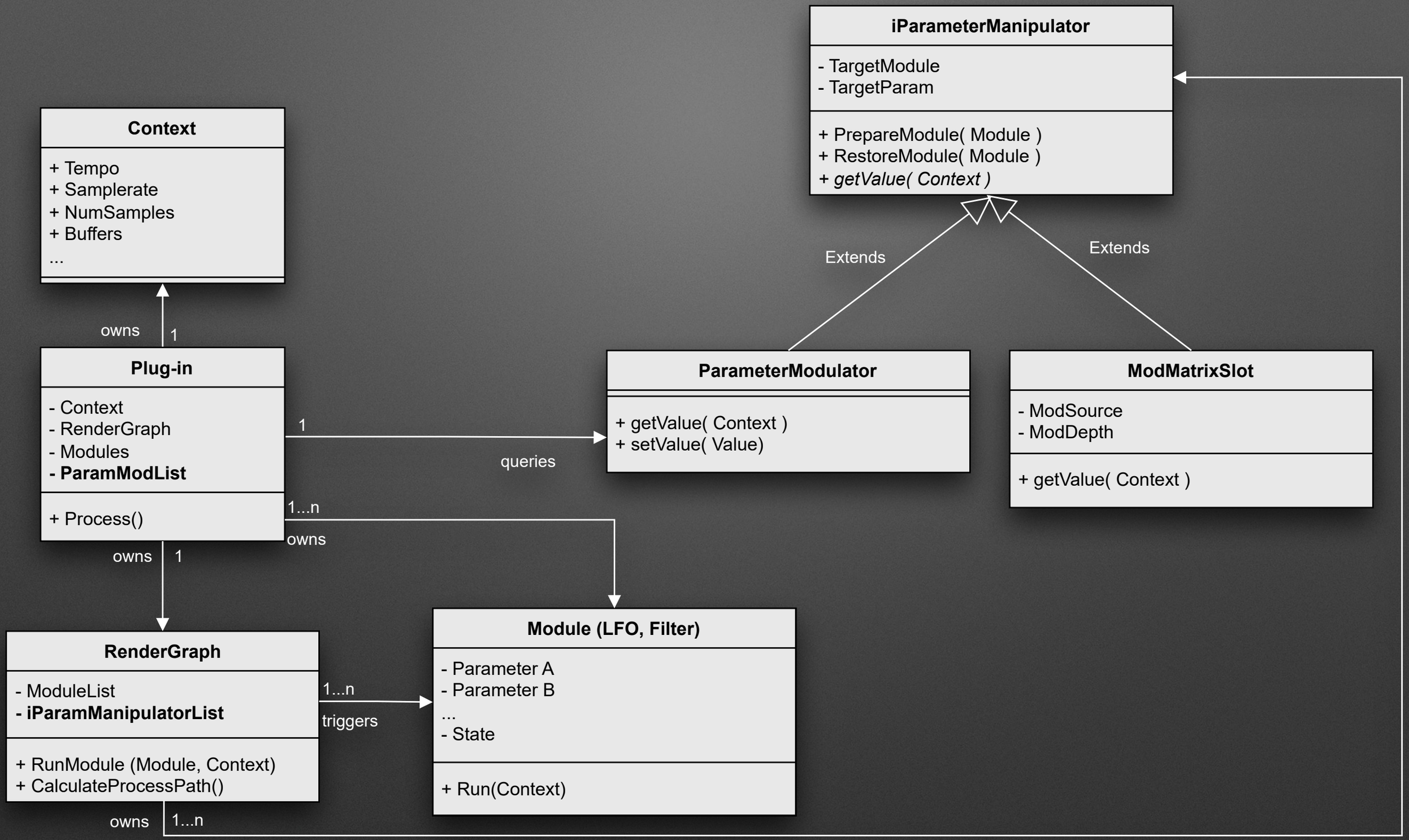
```
Context C = PlugIn->getContext()

for each Module in Modules:
{
        for each ModMatrixSlot in ModMatrixSlots
        {
                if( ModMatrixSlot->TargetModule == Module )
                {
                    // temporarily change ParameterValue
                    ModMatrixSlot->PrepareModule( Module, Context )
                }
        }

        Module->Run( Context )

        // iterate in reverse order, so we can stack multiple Slots

        for each ModMatrixSlot in reverse( ModMatrixSlots )
        {
                if( ModMatrixSlot->TargetModule == Module )
                {
                    // write back previous ParameterValue
                    ModMatrixSlot->RestoreModule( Module, Context )
                }
        }
}
```


iParameterManipulator
- TargetModule
- TargetParam
+ PrepareModule(Module)
+ RestoreModule(Module)
+ getValue(Context)
Context
+ Tempo
+ Samplerate
+ NumSamples
+ Buffers
...
Extends
Extends
owns
1
Plug-in
- Context
- RenderGraph
- Modules
- ParamModList
+ Process()
1
queries
ParameterModulator
+ getValue(Context)
+ setValue(Value)
ModMatrixSlot
- ModSource
- ModDepth
+ getValue(Context)
1...n
owns
owns
1
RenderGraph
- ModuleList
- iParamManipulatorList
+ RunModule (Module, Context)
+ CalculateProcessPath()
1...n
triggers
Module (LFO, Filter)
- Parameter A
- Parameter B
...
- State
+ Run(Context)
owns
1...n

pseudo code

```
Context C = PlugIn->getContext()

for each Module in Modules:
{
        for each PManipulator in ParameterManipulators
        {
                if( PManipulator->TargetModule == Module )
                {
                    // temporarily change ParameterValue
                    PManipulator->PrepareModule( Module, Context )
                }
        }

        Module->Run( Context )

        // iterate in reverse order, so we can stack multiple Slots

        for each PManipulator in reverse( ParameterManipulators )
        {
                if( PManipulator->TargetModule == Module )
                {
                    // write back previous ParameterValue
                    PManipulator->RestoreModule( Module, Context )
                }
        }
}
```

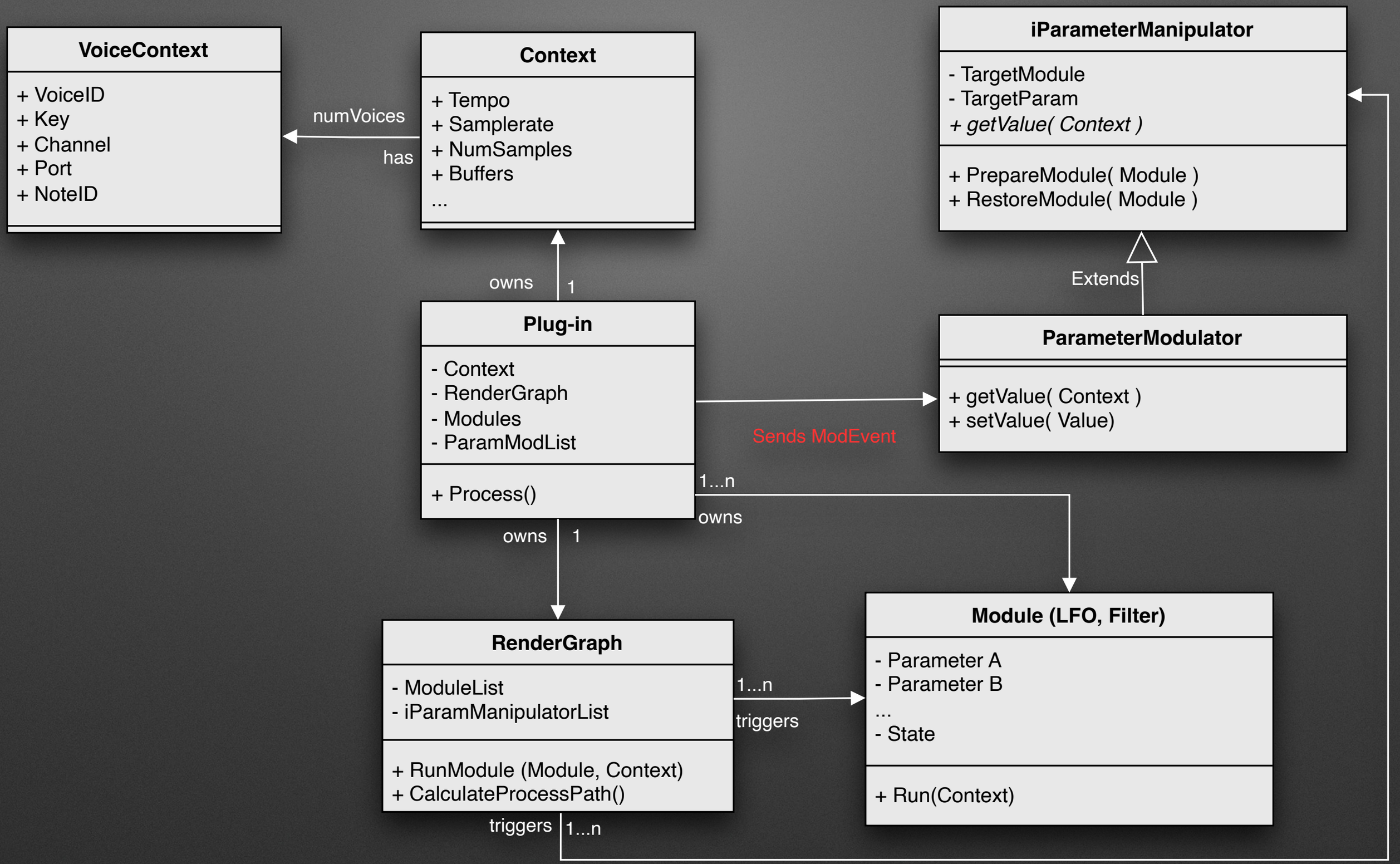
Polyphonic Parameter Modulation

- Uses same Event
- Targets Instrument Voices by
  - Port, Channel, Key (“PCK”)
  - Or NoteID
- Needs to relate Events to Voices over Time
- When Voices finish, outputs Event of Type `CLAP_NOTE_EVENT_END`

clap / include / clap / **events.h**

Code | Blame | 340 lines (289 loc) · 12.7 KB

```
234  typedef struct clap_event_param_mod {
235     clap_event_header_t header;
236
237     // target parameter
238     clap_id param_id; // @ref clap_param
239     void   *cookie;   // @ref clap_param
240
241     // target a specific note_id, port,
242     // -1 meaning wildcard, per the wild
243     int32_t note_id;
244     int16_t port_index;
245     int16_t channel;
246     int16_t key;
247
248     double amount; // modulation amount
249  } clap_event_param_mod_t;
250
```


VoiceContext
+ VoiceID
+ Key
+ Channel
+ Port
+ NoteID
numVoices
has
Context
+ Tempo
+ Samplerate
+ NumSamples
+ Buffers
...
iParameterManipulator
- TargetModule
- TargetParam
+ getValue(Context)
+ PrepareModule(Module)
+ RestoreModule(Module)
owns
1
Extends
Plug-in
- Context
- RenderGraph
- Modules
- ParamModList
+ Process()
Sends ModEvent
ParameterModulator
+ getValue(Context)
+ setValue(Value)
1...n
owns
owns
1
RenderGraph
- ModuleList
- iParamManipulatorList
+ RunModule (Module, Context)
+ CalculateProcessPath()
1...n
triggers
Module (LFO, Filter)
- Parameter A
- Parameter B
...
- State
+ Run(Context)
triggers
1...n

ModEvent

+ ParameterID
+ Value
+ Key (-1 = any)
+ Channel (-1 = any
+ Port (-1 = any)
+ NoteID (-1 = any)

iParameterManipulator

- TargetModule
- TargetParam
+ getValue(Context)

+ PrepareModule(Module)
+ RestoreModule(Module)

Stores
and Updates

1...n

Extends

ModEventTracker

- ModEventPool

+ setModEvent(ModEvent)
+ getModEvent(ParameterID, Key, Channel, Port,NoteID)

1

queries

ParameterModulator

+ getValue(Context)

Plug-in

- Context
- RenderGraph
- Modules
- ParamModList

+ Process()

Sends ModEvent

passes Note Information via Context

Concluding…

- We did not have to change anything in DSP Modules
- Just added CLAP Queue and extended ModMatrix
- No changes in UI, no Parameters added or modded
- Mod Event Tracker was reused for NoteExpressions

Q & A

Alexandre Bique & Urs Heckmann