PiPo

Plugin Interface for (Afferent Stream) Processing Objects

Norbert Schnell
IMTR, Real-Time Musical Interactions
IRCAM – Centre Pompidou
03/04/2013

Context

Experimentation and Prototyping

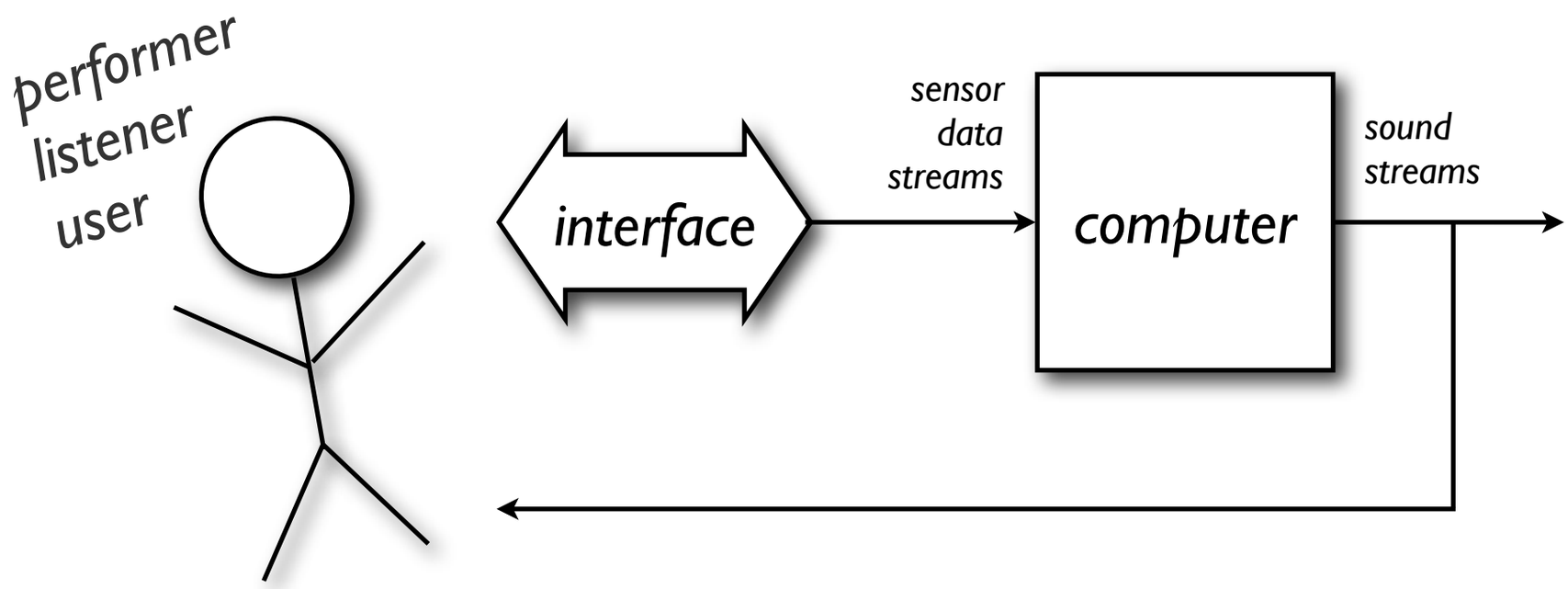
- We don't know exactly what we are doing
- We need everything *ready-at-hand*

  ... *in* and *off* Max (*MuBu for Max*, *IAE* on iOS)
- We have a lot of code to plug-in

  ... and the *simpler* the *better*.

Context

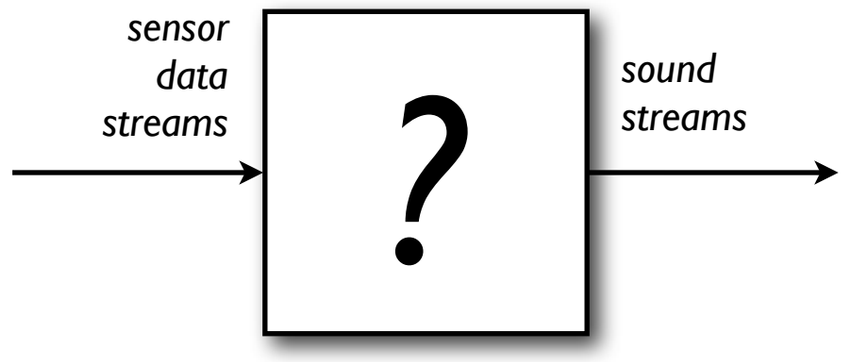
Interactive Audio Systems



Context

Interactive Audio Systems

our daily

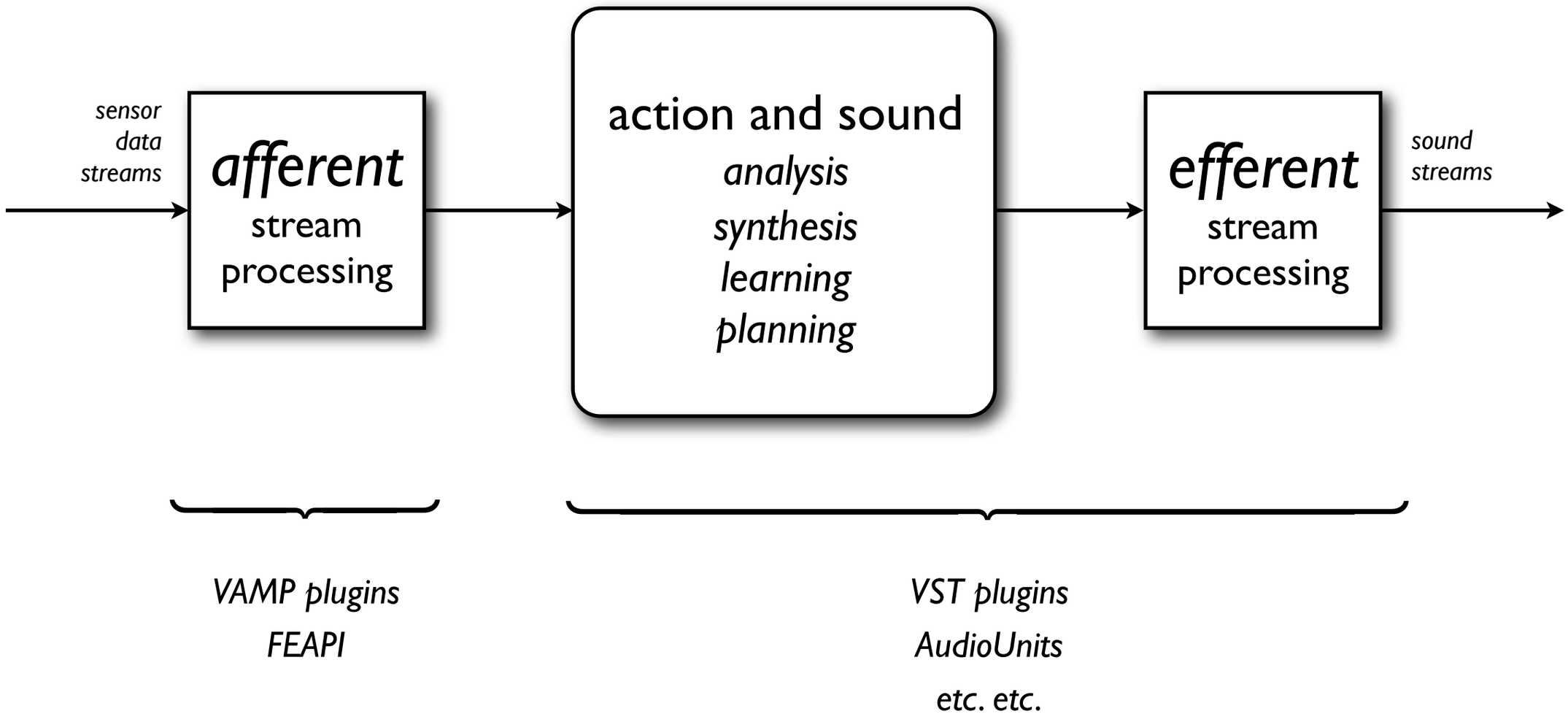


Context

Interactive Audio Systems

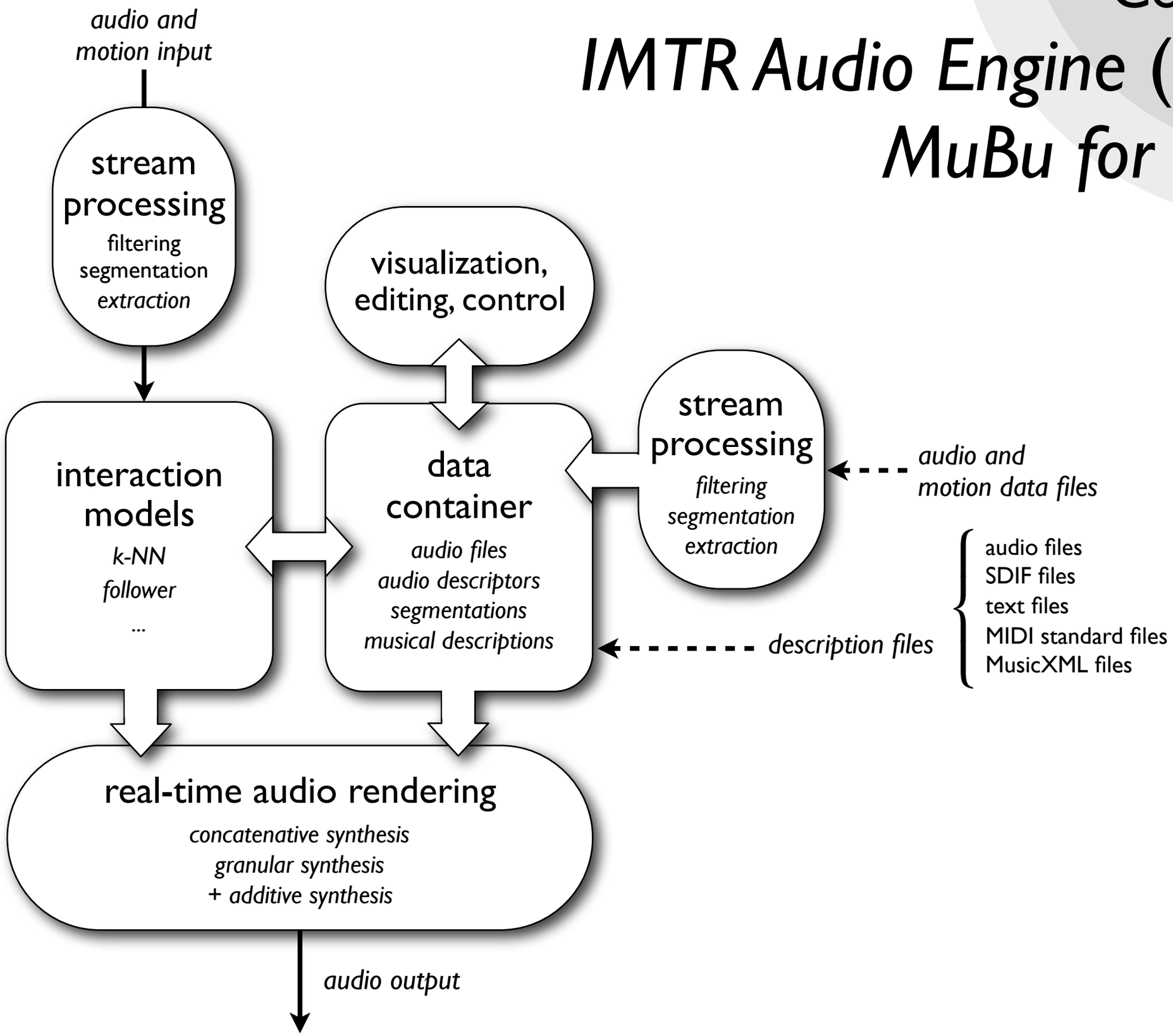
Context

Interactive Audio Systems



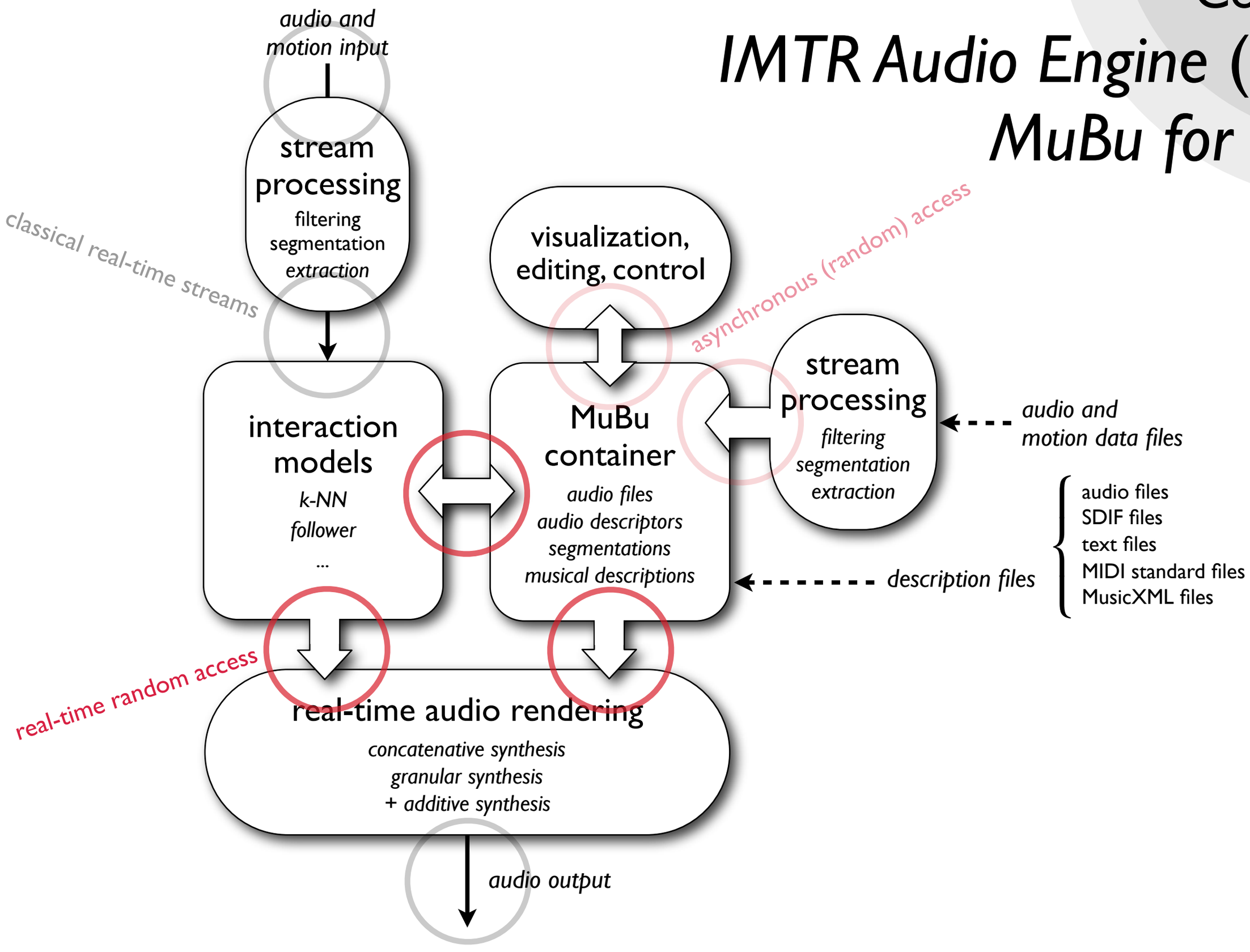
Context

IMTR Audio Engine (IAE)
MuBu for Max



Context

IMTR Audio Engine (IAE)
MuBu for Max



Summary

Afferent Streams

- *Real-Time* and *offline* processing
- *Synchronous* and *asynchronous* processing
- Multi-*dimensional* / *-modal* / *-format*
- Data *conditioning*, *reduction*, *annotation*
  - *filtering*
  - *extraction*
  - *segmentation*
  - *temporal modeling*
  - *classification*

What is PiPo?

- Very compact formalization of streams
- Module API (C++)
- Experimental host integration (*Max* and *IAE*)
- First set of modules
- First applications (*MuBu for Max*, *VoiceFollower*, iOS proto)

PiPo *Constraints* and *Motivations*

- Extremely simple framework (in a single .h !!!)
- Easily *integrate* modules into applications
- Easily *reuse* code (cf. *openFrameworks*)
- Inhomogeneous module *size* and *granularity*
  - elementary (e.g. *slice*, *fft*, *bands*, *dct*, *scale*, *sum*)
  - compact (e.g. *mel*, *mfcc*, *loudseg*)
  - integrated (i.e. extract + filter + segment + temporal modeling + classify)
- Integrate existing code (*FTM & Co.*, *MnM*, *etc.*)

PiPo *Formalization* (roughly *SDIF* inspired)

- *Chain* or *graph* of modules
- *One* in, *one* out, *multiple* branches
- Two mutually exclusive phases of streaming
  1. (setup stream): propagate *stream attributes*
  2. (real-time processing): propagate *frames*
- *Stream* attributes
  - stream *timing* (*time-tagged*, *sample rate* and *offset*, ...)
  - frame *dimensions* (scalars/vectors/2dim-matrices, ...)
- *Module* attributes (*real-time* or require *setup*)

PiPo *Implementation*

PiPo.h

```
class PiPo
{
private:
   std::list<PiPo *> receivers; /**< list of receivers */
   std::vector<Attr *> attrs; /**< list of attributes */

public:
   PiPo(PiPo *receiver = NULL);
   ~PiPo(void);

   // make connections
   virtual void setReceiver(PiPo *receiver, bool add = false);

   // setup module and propagate stream attributes
   virtual int streamAttributes(bool hasTimeTags, double rate, double offset,
                                unsigned int width, unsigned int size,
                                const char **labels, bool hasVarSize,
                                double domain, unsigned int maxFrames);

   // signal stream attributes changed
   virtual int streamAttributesChanged(unsigned int unitId = 0);

   // reset streaming, propagate frames, finalize streaming
   virtual int reset(void);
   virtual int frames(double time, double weight, PiPoValue *values, unsigned int size, unsigned int num);
   virtual int finalize(double inputEnd);

...

// attribute base class, template, and specializations
```

PiPo Module *PiPoScale*

```
#include "PiPo.h"

class PiPoScale : public PiPo
{
private:
  ... // internal stuff

public:
  enum ScaleFun { ScaleLin, ScaleLog, ScaleExp };

  PiPoVarSizeAttr<double> inMin;
  PiPoVarSizeAttr<double> inMax;
  PiPoVarSizeAttr<double> outMin;
  PiPoVarSizeAttr<double> outMax;
  PiPoScalarAttr<bool> clip;
  PiPoScalarAttr<PiPo::Enumerate> func;
  PiPoScalarAttr<double> base;

  PiPoScale(PiPo *receiver = NULL) :
    PiPo(receiver),
    inMin(this, "inmin", "Input Minimum", true),
    inMax(this, "inmax", "Input Maximum", true),
    outMin(this, "outmin", "Output Minimum", true),
    outMax(this, "outmax", "Output Maximum", true),
    clip(this, "clip", "Clip Values", false, false),
    func(this, "func", "Scaling Function", true, ScaleLin),
    base(this, "base", "Scaling Base", true, 1.0),
    ...
{
    ...

    this->func.addEnumItem("lin", "Linear scaling");
    this->func.addEnumItem("log", "Logarithmic scaling");
    this->func.addEnumItem("exp", "Exponential scaling");
}
```

module attribute declaration

module attribute instantiation

func attribute enum declaration

PiPo Module *PiPoOnseg*

```
#include "PiPo.h"

class PiPoOnseg : public PiPo
{
private:
  ... // internal stuff

public:
  PiPoScalarAttr<int> fltsize;
  PiPoScalarAttr<double> threshold;
  PiPoScalarAttr<double> mininter;

  PiPoOnseg(PiPo *receiver = NULL) :
    PiPo(receiver),
    fltsize(this, "filtersize", "Onset Filter Size", true, 3),
    threshold(this, "threshold", "Onset Threshold", false, 5),
    mininter(this, "mininter", "Minimum Onset Interval", false, 50.0)
  {
    ...
  }
```

module attribute declaration

module attribute instantiation

Propagate *Stream Attributes*

```
int PiPoOnseg::streamAttributes(bool hasTimeTags, double rate, double offset,
                                unsigned int width, unsigned int size, const char **labels, bool hasVarSize,
                                double domain, unsigned int maxFrames)
{
  unsigned int filterSize = this->fltsize.get();
  unsigned int inputSize = width * size;

  this->offset = -1000.0 / rate; // we are one frame off

  if(filterSize < 1)
    filterSize = 1;

  if(filterSize != this->filterSize || inputSize != this->inputSize)
  {
    // configure internal state
    this->buffer.resize(inputSize, filterSize);
    this->temp.resize(inputSize * filterSize);
    this->lastFrame.resize(inputSize);

    this->filterSize = filterSize;
    this->inputSize = inputSize;
  }

  return propagateStreamAttributes(true, rate, 0.0, 1, 1, NULL, false, 0.0, 1);
}
```

stream attribute propagation

Propagate *Frames*

```
int PiPoOnseg::frames(double time, double weight, PiPoValue *values, unsigned int size, unsigned int num)
{
  double onsetThreshold = this->threshold.get();
  double minimumInterval = this->mininter.get();

  for(unsigned int i = 0; i < num; i++) // iterate over input frames
  {
    PiPoValue onset;
    bool isOnset = frameIsOnset(values, size, onsetThreshold, minimumInterval, onset);

    if(isOnset && !this->lastWasOnset && time >= this->lastOnsetTime + minimumInterval)
    {
      // we have an onset here
      int ret = propagateFrames(this->offset + time, weight, &onset, 1, 1); // propagate output frames

      if(ret != 0)
        return ret;

      this->lastOnsetTime = time;
    }

    this->lastWasOnset = onset;

    values += size;
  }

  return 0;
}
```

Reset and *Finalize* Streaming

```
int PiPoOnseg::reset(void) // reset module before streaming
{
  this->buffer.reset();

  this->lastWasOnset = false;
  this->lastOnsetTime = -DBL_MAX;

  return propagateReset();
};
```

```
int PiPoOnseg::finalize(double inputEnd) // finalize stream at end (if any)
{
  return propagateFinalize(inputEnd);
}
```

Compose PiPo Modules

PiPoMel.h

```
#include "PiPoSlice.h"
#include "PiPoFft.h"
#include "PiPoBands.h"

class PiPoMel: public PiPoSlice // inherit first PiPo in the chain (mel = slice + fft + bands)
{
private:
  PiPoFft fft; // internal PiPo (2nd in chain)
  PiPoBands bands; // internal PiPo (3rd in chain)

public:
  PiPoMel(PiPo *receiver = NULL) :
    PiPoSlice(&this->fft), // instantiate 1st and connect to 2nd
    fft(&this->bands), // instantiate 2nd and connect to 3rd
    bands(receiver) // instantiate 3rd and connect to receiver
  {
    // steal attributes from internal PiPos
    this->addAttr(this, "windsize", "FFT Window Size", &this->size, true);
    this->addAttr(this, "hopsize", "FFT Hop Size", &this->hop);
    this->addAttr(this, "numbands", "Number Of Bands", &this->bands.num);
    this->addAttr(this, "log", "Logarithmic Scale Output", &this->bands.log);

    // set internal PiPo attributes
    this->wind.set(PiPoSlice::BlackmanWindow);
    this->norm.set(PiPoSlice::PowerNorm);
    this->fft.mode.set(PiPoFft::PowerFft);
    this->bands.mode.set(PiPoBands::MelBands);
    this->bands.log.set(false);
  }

  // overload set receiver to connect to last PiPo (bands)
  void setReceiver(PiPo *receiver, bool add) { this->bands.setReceiver(receiver, add); };
};
```

Max Plugins (Externals)

```
#include "PiPoScale.h"
#include "MaxPiPo.h"

// generate pipo.scale (“NOBOX”) Max external
PIPO_MAX_CLASS("scale", PiPoScale);
```

```
#include "PiPoMel.h"
#include "MaxPiPo.h"

// generate pipo.mel (“NOBOX”) Max external
PIPO_MAX_CLASS("mel", PiPoMel);
```

Available PiPo Hosts (3/4/2013)

- Max externals *pipo* and *pipo~*
- Max external *mubu.process* (*MuBu for Max*)
- IAE method *applyPiPo()* *(Max, Unity 3D, iOS proto)*

Available PiPo Modules (3/4/2013)

- *psy* ... pitch synchronous yin
- *mvavrg, median, biquad* ... simple filters
- *scale, sum* ... misc modules
- *slice, fft, bands, dct* ... spectral description building blocks
- *mel, mfcc* ... spectral description
- *onseg* ... onset detector (yet to be released)
- *ircamdescriptor* ... similar to *ircamdescriptor~* (yet to be released)

Pi... Po... Perspectives (3/4/2013)

- *More* modules
- Integration of *temporal modeling*
- Consolidate API
- Share PiPo *framework*
- Share PiPo *modules*

http://imtr.ircam.fr/imtr/PiPo