CLAM:
C++ Library for Audio and Music

Introducing the framework

Goals (I)

- Original Goal: "To offer a complete, flexible and platform independent Sound Analysis/Synthesis C++ library to meet current and future needs of all MTG projects."

- Complete: should include all utilities needed in a Sound Processing Project (input/output, processing, storage, display...)
- Flexible: Easy to use and adapt to any kind of need.
- Platform Independent: Compile under Unix, Windows and Mac platforms.

Goals (II)

- The original goals have slightly changed:
  - CLAM is part of the European project AGNULA (A GNU Linux Audio Distribution): Demudi and Rehmudi distributions.
  - CLAM is free software (GPL)
  - CLAM is public.

Fact Sheet

- Started in October 2000
- Currently has more than 250 C++ classes (50.000 loc) compiled and tested under Linux, Windows and partially under MacOS.
- 8 people are working on the core of CLAM
  - Xavier Amatriain
  - Maarten de Boer
  - Pau Arumi
  - David Garcia
  - Miquel Ramírez
  - Xavier Rubio, Albert Mora, Sandra Gilabert
- CLAM has already been used for a number of internal projects: near lossless time-scaling, saxo synthesis, content-analysis and Mpeg7 description, real-time performance,...
- CLAM is the base for most Final Studies projects that are done in the MTG.

Why CLAM is different to "anything" else (?)

- Basic difference between spectral processing vs. Time-domain processing
- Buffer processing vs. sample by sample processing
- Multiple kinds of Processing Data can travel through the signal path
- We do not have a single Signal class
- Objects may need to process different amounts of data
- There is not a unique definition of data-chunk
- Cross-platform
- Two-folded usage (library and application)

Projects related to CLAM

- OSW: Open Sound World
- JMAX
- PD
- SoundClass
- AudioMulch
- ...

CLAM's two modes

- Supervised mode (application)
  - Under development
- Non-supervised mode (library)
  - Has already seen 2 internal releases and is the base for the public version.
  - A number of internal projects have already used it: Time Machine, SALTO, CUIDADO, Voice Processor
  - Many of the design decisions are driven by supervised mode compatibility

The non-supervised mode

A C++ library for audio processing

Recommended previous skills

- OO Analysis and Design
  - You have to be able to identify classes for an application and convert a process thread into a class diagram
  - UML is recommended…
- You need some previous C++ knowledge.

Dynamic Types (i)

- Motivation: in C++ (and in most OO languages) it is not possible to instantiate/deinstantiate attributes in run-time.
- Dynamic Types are the base for all CLAM Processing Data and configurations.
- Goals
  - Enable the creation of new classes that comply to CLAM specifications

Dynamic Types (ii)

- Offer a tree-like homogeneus structure and tools to navigate it.
- A Dynamic Type is like a normal C++ class but it allows to work with non-instantiated attributes.
  - These attributes can be added or removes at run-time.
- Furthermore, each dynamic attributa has a homogeneous interface (Add, Remove, Set, Get) that is implemented automatically.

Dynamic Types (iii)

- The implementation is based on precompiler macros and templates.
- To define a new class, you have to use some easy macros.
  - Example:

```
class Note : public DynamicType
{
public:
        DYNAMIC_TYPE (Note,4);
        DYN_ATTRIBUTE (0, public, int, NSines);
        DYN_ATTRIBUTE (1, public, Array<Sines>, Sines)
        DYN_ATTRIBUTE (2, public, float, Pitch);
        DYN_ATTRIBUTE (3, public ADSR, myADSR);
};
```

Dynamic Types (iv)

- Instantiating attributes
  - When a Dynamic Type is instantiated, its attributes are not necessarily instantiated (only those that have been instatiated in the DefaultInit() method).

    Note myNote; (Will only instantiate those attributes instantiated in Note::DefaultInit())
  - To instantiate attributes by hand

    myNote.AddPitch();
    myNote.UpdateData(); (only once for a set of Add/Replace operations)
- Using a dynamic type

  myNote.SetPitch(440.2);
  float pitch=myNote.GetPitch();

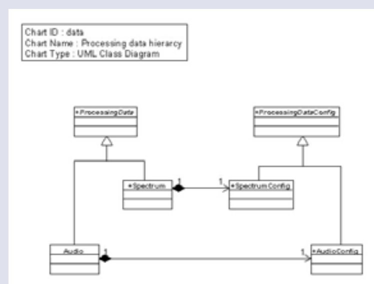
Processing Data (i)

- All data in the signal path must be encapsulated as a Processing Data Class
- Processing Data Classes are Dynamic Types
- Most of their interface need not be implemented (setters and getters are automatically generated through DT)
- Processing data persistency is accomplished through direct (and automatic) XML mapping.
- Inputs and outputs to a Processing object have to be Processing Data.

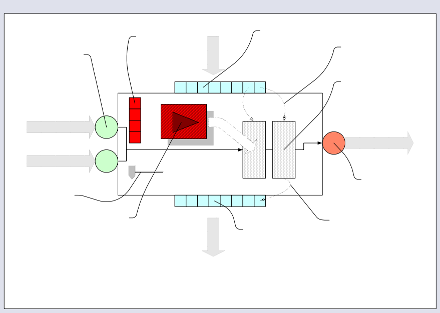
Processing Data (ii)

- Complex Processing Data classes (such as spectrum) may have an associated configuration class.
- The default constructor calls the DefaultInit() method.
  - This is where default attributes should be instantiated.

Processing Data (iii)



Processing (i)



Processing (ii)

- All processing in CLAM must be performed inside a Processing class.
- In the non-supervised mode, data is input as arguments of the Do(...) method.
  - A more complex/efficient Port infrastructure may also be used.
- The Do(...) method (any of its overloads) is the only method that is called from the external processing loop.
- This method is executed at the processing rate.

Processing (iii)

- Control signals are treated in a very different way:
  - Controls generate “events” only when their state or value is modified.
  - These events travel to input controls belonging to another processing object (these controls have been previously connected).
  - Processing objects can publish methods that are used as functions called by input controls.
  - Processing objects can generate events for their output controls during the Do(...) execution.

Processing (iv)

- Processing classes have always an associated configuration class.
  - It is a related class where configuration parameters are stored.
  - It can also hold initial values for the controls.
  - A configuration parametre can only be modified if the processing object is not in “running” status.

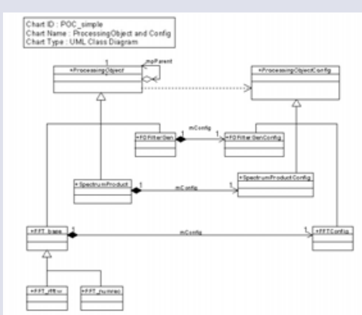
Processing (v)

- Processing objects may be in one of the following states: *running, unconfigured, disabled o ready*

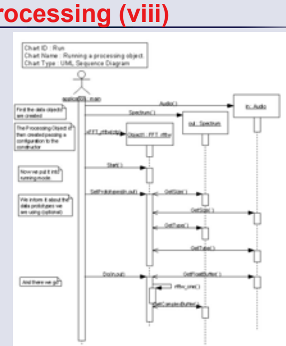
Processing (vi)

- Inputs and outputs are always Processing Data (dynamic types)
  - Processing objects have to check for the consistency of the data that is passed as arguments of the Do(...) methods.
  - This may be done every time the method is called.
  - An alternative (and recommended) way is to use prototypes.
    - Once a prototype is configured, the Processing object “believes” that everything that arrives follows the prototipe (or else the system will crash!).

Processing (vii)



Processing (viii)



XML Interface (i)

- Goals
  - Implement data persistency.
  - Offer a way to store an object in an appropiate generic format (XML).
  - Offer XML output for any class with a minimum programmer effort.
  - Offer automatic ways to make XML representations out of classes.
  - Allow integration of other formats (SDIF, binary...)
- Dynamic Types have an automatically derived XML interface (Store/Load).
- So, Processing Data classes and configurations also.

XML Interface (iii)


```
-<Spectrum>
  -<prConfig>
    <Scale>Linear</Scale>
  <SpectralRange>4000</SpectralRange>
  <Size>513</Size>
  <Type>MagPhase Complex</Type>
  </prConfig>
  -<MagBuffer>
  <content>3.98157 4.02727 4.16642 4.40572 4.75821 5.24668 5.90992 6.81515 8.08461 9.96051
    12.9877 18.6098 31.1381 60.5695 98.2945 89.1955 54.8392 30.1422 19.2311 14.0992 11.235
    9.40867 8.13581 </content>
  </MagBuffer>
  -<PhaseBuffer>
    <content>3.14159 3.03485 2.93081 2.83178 2.73945 2.65478 2.57805 2.50887 2.44616 2.38745
    2.32608 2.23915 2.04041 1.47977 0.115056 3.09922 -1.87873 -1.376 -1.21958 -1.17472 -
    1.16297 -1.16238 -1.16613 </content>
  </PhaseBuffer>
  </Spectrum>
```


Visualization

- CLAM offers an architecture to develop a user interface.
- It is a number of classes that implement a MVC-like architecture.
- Furthermore, thera are also ready-to-use tools
  - Views for the more important processing data
  - Debugging tools (Snapshots)

Audio I/O (i)

- The main class for audio input/ouput is the AudioManager:
  - This class is in charge of all the administrative tasks related with *stream* creation and initialization, using the AudioDevice class.
- The first thing that you have to do to use audio is to create an AudioManager object (singleton) that will be called by all other audio I/O objects.
- Then you can use the AudioIn and AudioOut classes to read or write audio to the soundcard.
  - These objects are created from an AudioIOConfig configuration where you specify the device, the channel and the sampling rate.
  - These objects process mono channels.
- To specify a device, you must use a string with the following syntax:

  "ARCHITECTURE:DEVICE"

Audio I/O (ii)

- Until this moment we have implemented the alsa architecture and directx for windows (using PortAudio, RTAudio or DirectX directly).
  - The devices that are available (depending on the hardware and the system configuration) can be consulted using the AudioDeviceList class.
  - But, if you do not specify the device or you use the "default:default" string, the AudioManager will choose the most convenient.
- You can specify what channel you want for each AudioIn or AudioOut. The AudioManager will use this information to initialize its internal management. Usually, 0 is used for L and 1 for R.

Audio I/O (iii)

- Example:

```
AudioManager audioManager;

inCfgL.SetName("left in");
inCfgL.SetChannelID(0);

inCfgR.SetName("right in");
inCfgR.SetChannelID(1);

AudioIn inL(inCfgL);
AudioIn inR(inCfgR);
```

Audio I/O: file

- The input/ouput of audio files is performed using the AudioFileIn and AudioFileOut Processing classes.
- At this moement, we only support raw, aiff and wav formats.
- But, opposite to most existing libraries, it allows for concurrent reading/writing.

MIDI IN

- At this moment we only have MIDI Input for Linux and Windows (based in PortMIDI).
- The architecture is very similar to the Audio I/O: you also have a MIDIManager.
- There is a MIDIIn class and a derived MIDIInControl class that is used to convert input MIDI messages to CLAM controls.
- The MIDIInConfig class has 3 parametres that specify what MIDI messages will be filtered to a MIDIIn object.
    - ChannelMask (bit mask)

      ```
      cfg.SetChannelMask( MIDI::ChannelMask(1) | MIDI::ChannelMask(2) );
      ```

    - MessageMask (bit mask)

      ```
      cfg.SetChannelMask(MIDI::MessageMask(MIDI::eNoteOff) |
      MIDI::MessageMask(MIDI::eNoteOn) );
      ```

    - Filter (filter to apply to the second byte of the MIDI message)

MIDI IN

- The MIDIInControl class implements MIDIIn with one ore more output controls (the number of outputs depends on the type of filter that is used)
- Controls are generated for each MIDI message that is received.
- If, for example, you configure a MIDIInControl for eNoteOn messages, you will obtain two OutControls (one for the key and the other for velocity).

Tools and frameworks used in CLAM (i)

- Programming language: C++
    - flexibility
    - efficiency
    - standard vs. propietary language
- Programming frameworks
    - Windows: Visual C++ (6.0 or 7.1 Everett), Rational Purify, Rational Distiller.
    - Linux: g++, kde and other GNU tools
    - Mac: CodeWarrior, g++

Tools and frameworks used in CLAM (ii)

- CVS: System for version control in collaborative code development (Windows: WinCVS).
- Mantis: Bug managing system through a web interface
- Doxyen: Program that generates html documentation pages automatically from javadoc comments inserted in the code.
- Mailing-list

External Libraries

- FFTW (FFT)
- Xerces (to parse XML using the DOM API)
- FLTK (for GUI)
- PTHREADS (for cross-platform multithreading handling)
- PortMidi (Windows MIDI)
- DirectXSDK, RtAudio and PortAudio (Windows Audio)

Conclusions

- Although there are still a lot of things to do, CLAM is already an interesting development framework that can be used for developing efficient and robust audio applications and research works.
- In the CLAM Tutorial, CLAM is used in a very specific way (Spectral Analysis/Synthesis).
  - It does not deal with Audio I/O, MIDI or real time processing.
  - If you are interested in those subjects you may work on the different examples availabla (SALTO and SpectralDelay).