

Disclaimer: this pdf is just the content of the slides with no format at all. I'm using the moin-wiki slides plugin for the real slides.

Introduction (1)

Pau Arumi – parumi@iua.upf.es

CLAM: C++ Library for Audio and Music

- A Framework for research and application development
- Offers a **conceptual model** and tools for analysis, synthesis and transformations
- **Cross-platform** : GNU/Linux, Mac OSX, Windows

Intro (2) Initial Goals

- To meet the needs of all projects of the **Music Technology Group** (Universitat Pompeu Fabra)
- Deal with the **code-reuse problem** in the MTG.

Slightly changed goals

- Now the library is not seen as an internal tool for the MTG:
- GPL and public www.iua.upf.es/mtg/clam
- Agnula IST European project (some of the CLAM applications included in Demudi)

Developer team

- The same 3-4 core-developers have been working on it since 2001
 - Usually partial time dedication
-

What CLAM has to offer ? (1)

- Other similar environments exists: [OpenSoundWorld](#), Marsyas, [SndObj](#), Max, Pd, [SuperCollider...](#)
[1](#)
- CLAM feature-set makes it different:
 - **Object Oriented C++**
 - Framework Design Patterns, Design Patterns (GoF), C++ Idioms
 - Good development practices: Test Driven Development, Refactorings, Peer reviewing

- **Cross-platform** : GNU/Linux, Mac OSX, Windows
 - [1](#)
Study and comparison of most of them in X. Amatriain Phd Thesis.
-

What CLAM has to offer ? (2)

- ... Feature set:
 - **Efficient**: we took care in avoiding inefficiencies in the processing code
 - **Comprehensive** : not just processing but input/output, serialization services, data visualization, OSC control, etc.
 - **Extensible data types** : not just audio samples and spectrums.
 - **GPL** and public www.iaa.upf.es/mtg/clam
 - The framework can be used either as a regular C++ library or as a **prototyping tool**.
 - Comes with various usable (sample) applications.
-

CLAM as a framework

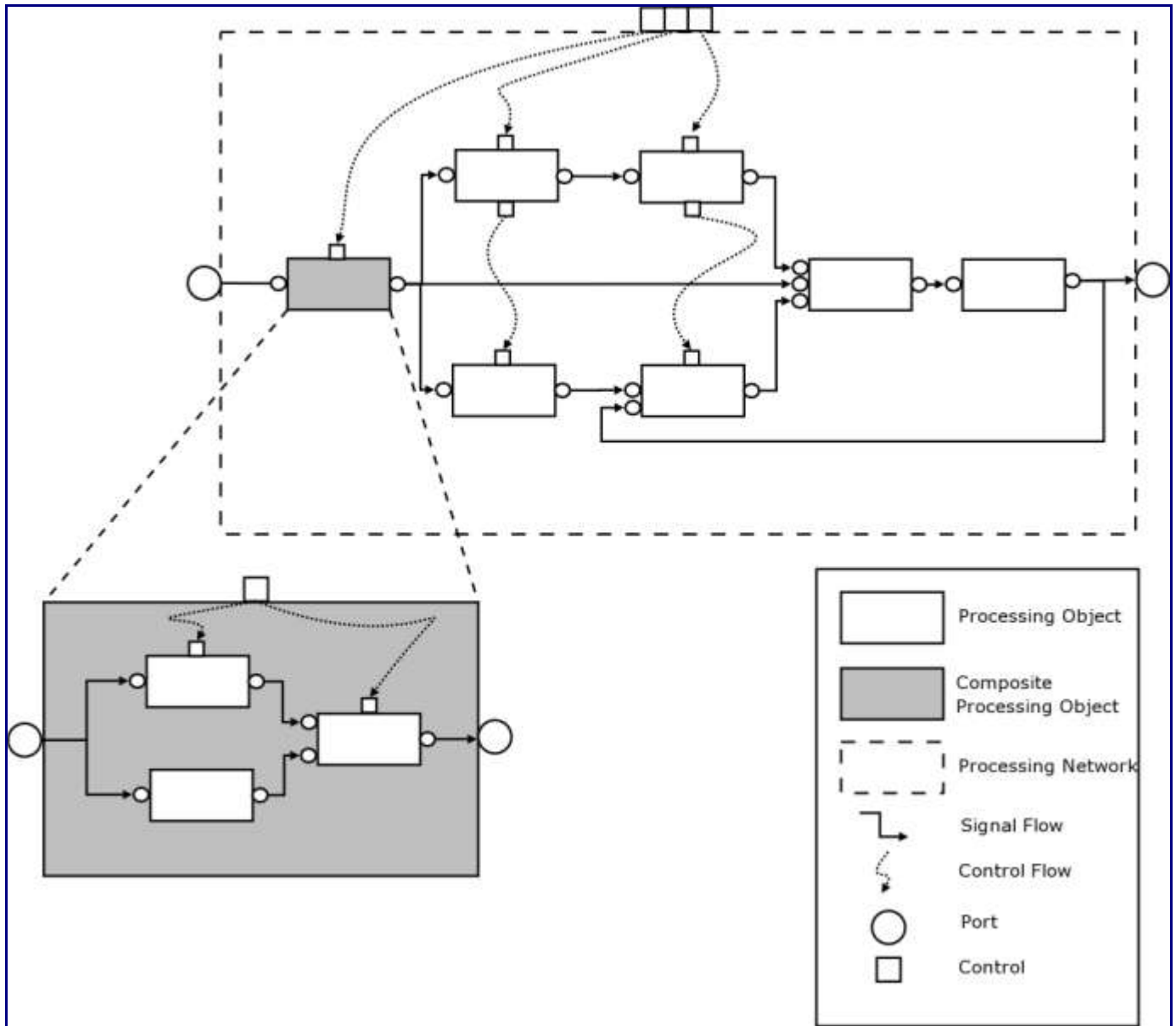
CLAM is an evolving framework. At this moment is both **white-box** and **black-box** [1](#)

- **white-box**
 - abstract classes can be easily derived to extend the components (processings and data).
- **black-box**
 - built-in components in the repositories can be connected with minimum programmer effort.
- [1](#)

Evolving Frameworks Roberts and Johnson, 1996

The CLAM model (1)

- Very related with the *Graphical Models of Computation*
- OO encapsulation of a graph for modelling DSP systems



Model (2). Two kinds of channels

Ports

- *Synchronous* : tokens are produced and consumed at a predictable rate.
 - *Pseudo-FIFO* queues

Controls

- *Asynchronous* : event driven

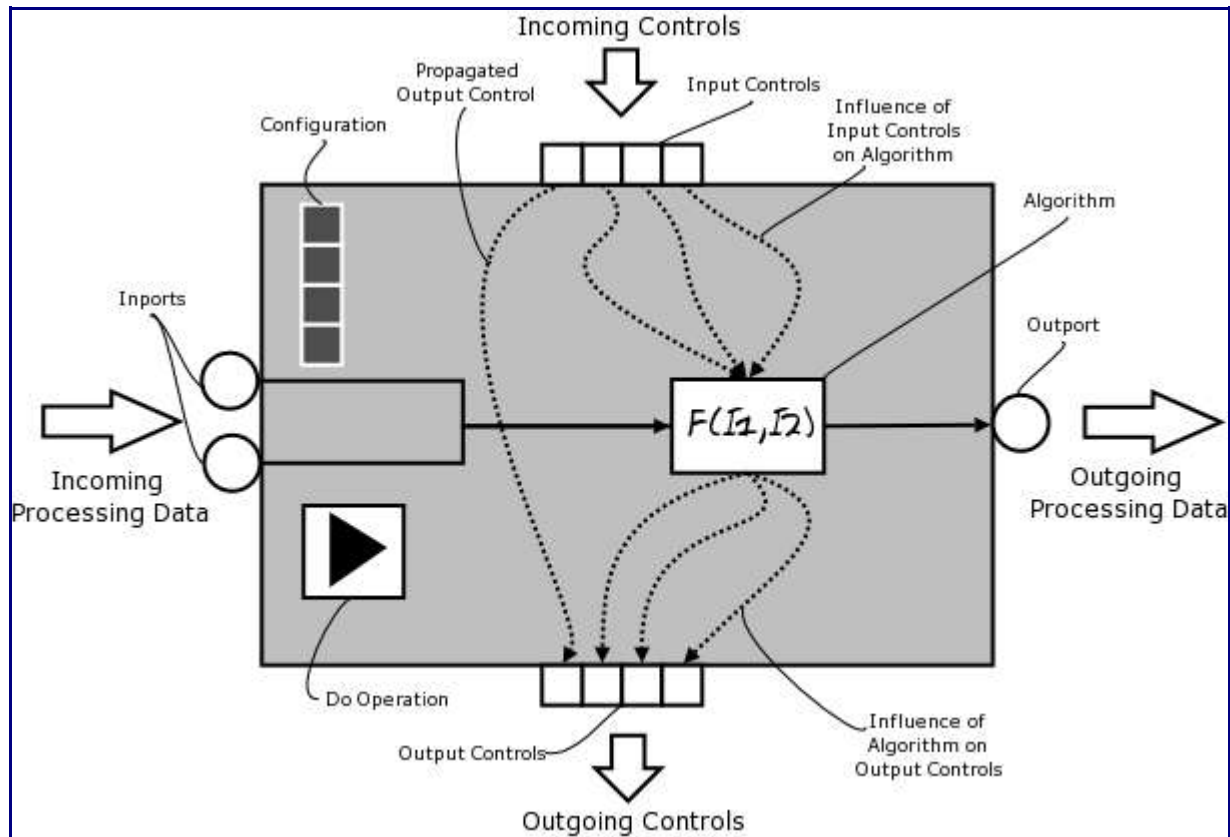
- may come in bursts
 - usually we only want to know the last event received before the firing of a processing
 - control data typically comes from a non processing thread
-

Model (3). Stream Nodes and Flow Control

Complex streaming flow

- One out port can feed *many in ports*
 - Different processings can consume and produce at *different rates* (num. tokens)
 - Only out-in ports of the *same type* they are connectable and regardless of the port *size*
 - [FlowControl](#) object is responsible of firing the processings avoiding:
 - firing a processing with not enough data in one of its in-ports
 - firing a processing with not enough space to write data in its out-port
 - [FlowControl](#) is subclassified to allow different strategies (*push, pull, static scheduling...*)
-

Model (3). Processings



Model (4). Configurations: why not just controls?

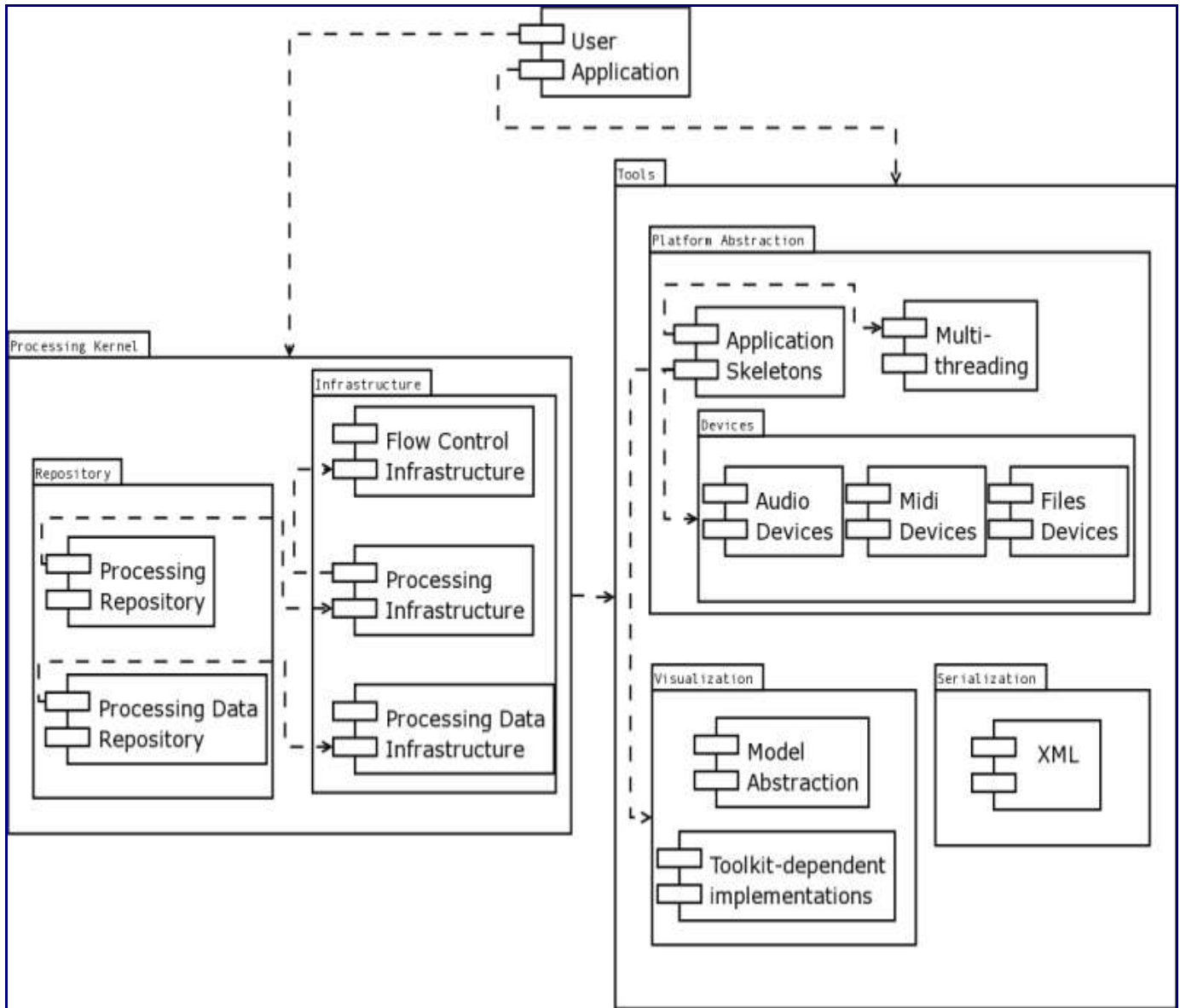
- Configurations are a different kind of parameters
- Dedicated to **expensive or structural** changes in the processing.
- Examples: Number of ports, FFT size
- Configurations are set when processing is not in *Running State*

Model (5). Composites: static vs dynamic

- It's possible to encapsulate various processing that works together in a composite processing.
- Enhances abstraction and reuse
- Inner ports and controls can be *published* to the outer composite
- Two kinds : **static** and **dynamic**
 - *Static* ones are hardcoded C++ classes.

- *Dynamic* ones are instances of the Network class.
- Trade-off between *efficiency vs. understandability*

Architectonic modules



Repositories

Bias to spectral processing

- *Processing Repository*

Analysis, ArithmeticOperators, AudioFileIO, AudioIO, Controls, Generators, MIDIIO, Plugins, SDIFIO, Synthesis, Transformations

- *Processing Data Repository*

Audio, Spectrum, SpectralPeaks, Envelop, Segment, Fundamental, Melody

- All processing data classes have homogeneous interface and built-in automatic XML serialization
-

Tools

- XML
 - GUI, support for FLTK and QT toolkits
 - *Plots* for each data type (useful for debugging)
 - Macro mechanism gives automatic XML and GUI support to processing data and processing configs.
 - Platform abstraction: Audio IO, MIDI IO, SDIF support
-

Levels of automation

1. **Library functions**
2. **Processing networks**
3. **Automatic processing networks**

(1) Library functions:

```
Audio audio;  
...  
Synth synth;  
...  
synth.Do(audio);
```

(2) Processing networks

```
AudioFileReader reader;  
SMSAnalysis analysis;  
ConnectPorts(reader, "Audio Out", analysis, "Audio In");  
...
```

```
reader.Do();
analysis.Do();
```

(3) Automatic processing networks

```
Network net;
... "Create" and "Connect" calls to the network
net.Do();
```

Integration with Linux audio infrastructure (1)

- **ALSA**
 - Audio IO, allowing low-latency real-time
 - Audio file libraries
 - Straight-forward to use. *libsndfile*, *libvorbis*, *libmad*, *libid3*
 - **OSC**
 - *oscpack*: small C++ library by Ross Bencina to be released soon.
-

Integration with Linux (2)

- **LADSPA**
 - Supported in two ways:
 - LADSPA-host processing class
 - Wrapper (template) class converts a CLAM processing class into a LADSPA plugin
 - To be done: a wrapper that converts the processing-network host in a LADSPA plugin.
 - **Jack**
 - One priority *to-do*: make the processing-network host jack-enabled
 - Our host now uses blocking I/O 😞
 - **DSSI**
 - Looks promising and fits very well together with our prototyping tool
 - (now only supports standalone apps.).
-

What CLAM can be used for ?

Have been used in several applications:

- **SMSTools**, an SMS analysis/synthesis graphical tool
 - **Salto**, a sax synthesizer
 - **Rappid**, a real-time processor used in live performances
 - **Annotator** (about to be released!)
 - Others: [Voice2Midi](#), time-stretch ...
-

Rappid Prototyping in CLAM

NetworkEditor: the CLAM visual builder tool

A three steps demo:

1. Building the processing network
 - With the [NetworkEditor](#)
 2. Designing the GUI
 - Using QT designer
 3. Running the prototype
 - Using an application with a **processing network host**
-

Conclusions and future work (1)

- We are tending to make CLAM more user oriented
 - More *black-box* framework with more *visual builder* tools

We are expecting to be able to increase the development effort.

Conclusions (2)

We are about to release 0.8

Lots of things to be done till CLAM 1.0 ¹:

- Network host with jack and DSSI support
- Split into several library binaries
- Finish the *feature-extraction sub-framework*
- **Simplify part of the code**
 - (specially related with processing data and configurations)

- Have working *nested networks*
 - [1](#)
see the routemap to 1.0 in the web
-

—
Questions ?

—
Thanks!