CLAM:
C++ Library for Audio and Music

Introduction

Goals (I)

- Initial Goal : “To offer a complete, flexible and cross-platform framework to support current and future needs for all projects developed in the MTG”

  - Complete: should include all general utilities needed in an audio processing project (input/output, processing, storage, graphical interface...)
  - Flexible: easy to use and adap to any necessity.
  - Cross-platform: should compile on Windows, GNU/Linux and Mac OSX

Goals (II)

- The initial goals have “slightly” changed:
  - CLAM is part of the AGNULA (A GNU Linux Audio Distribution) IST project: Demudi and Rehmudi distributions.
  - CLAM is Free Software (GPL)
  - CLAM is public

Factsheet

- Started in October 2000
- There are more than 250 C++ classes (50.000 loc), compiled under GNU/ Linux, Windows and OSX.
- 7 people work on the CLAM core:
  - Xavier Amatriain
  - Pau Arumi
  - David Garcia
  - Maarten de Boer
  - Miquel Ramírez
- CLAM has been used for a number of internal projects: time-stretching, real-time sax synthesis, content analysis and Mpeg7 description (CUIDADO), real-time audio effects.
- The students at the university also make extensive use of the framework.

Why is CLAM different to other frameworks?

- Basic difference between spectral and temporal domain processing
  - Buffer processing vs. sample-by-sample processing.
- There are different kinds of data travelling through a CLAM network
  - There is not a single “signal” class
- Objects can process different quantities of data
  - There is not a unique definition of data chunk.

Why is CLAM different to other frameworks?

- 100% Cross-platform
- It is a framework, not a library
- Two different working modes: framework and rapid-prototyping application.
- It is really Object-oriented
- It is efficient and can be used for real-time applications

Similar frameworks

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

The CLAM infrastructure

A C++ framework for audio and music processing

Dynamic Types

- Foundation: in C++ (and many OO languages) it is not possible to instantiate/deinstantiate object attributes on run-time.
- Dynamic Types are the base for Processing Data and configuration classes

Dynamic Types

- Goals
  - Make it easy to create new classes that conform to CLAM specifications
  - To offer a homogeneous and easy-to-navigate tree structure.
  - A Dynamic Type is like a regular C++ class but it allows to work with non-instatiated attributes.
    - These attributes can be added/removed on rum-time
  - Also, every attribute has a homogeneus interface (Add, Remove, Set, Get) that is automatically derived.

Dynamic Types

- Implementation based on pre-compiler macros and templates
  - To define a new class, some simple macros must be used.
  - 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

- Attribute instantiation
  - When a Dynamic Type is instantiated, its attributes are not all automatically instantiated: only those that are explicitly instantiated in the DefaultInit() operation.

```
Note myNote; (Will only have those attributes
instantiated in the Note::DefaultInit() method)
```

  - You can instantiate attributes by hand:

```
myNote.AddPitch();
myNote.UpdateData(); (only once for a set
of Add/Replace operations)
```

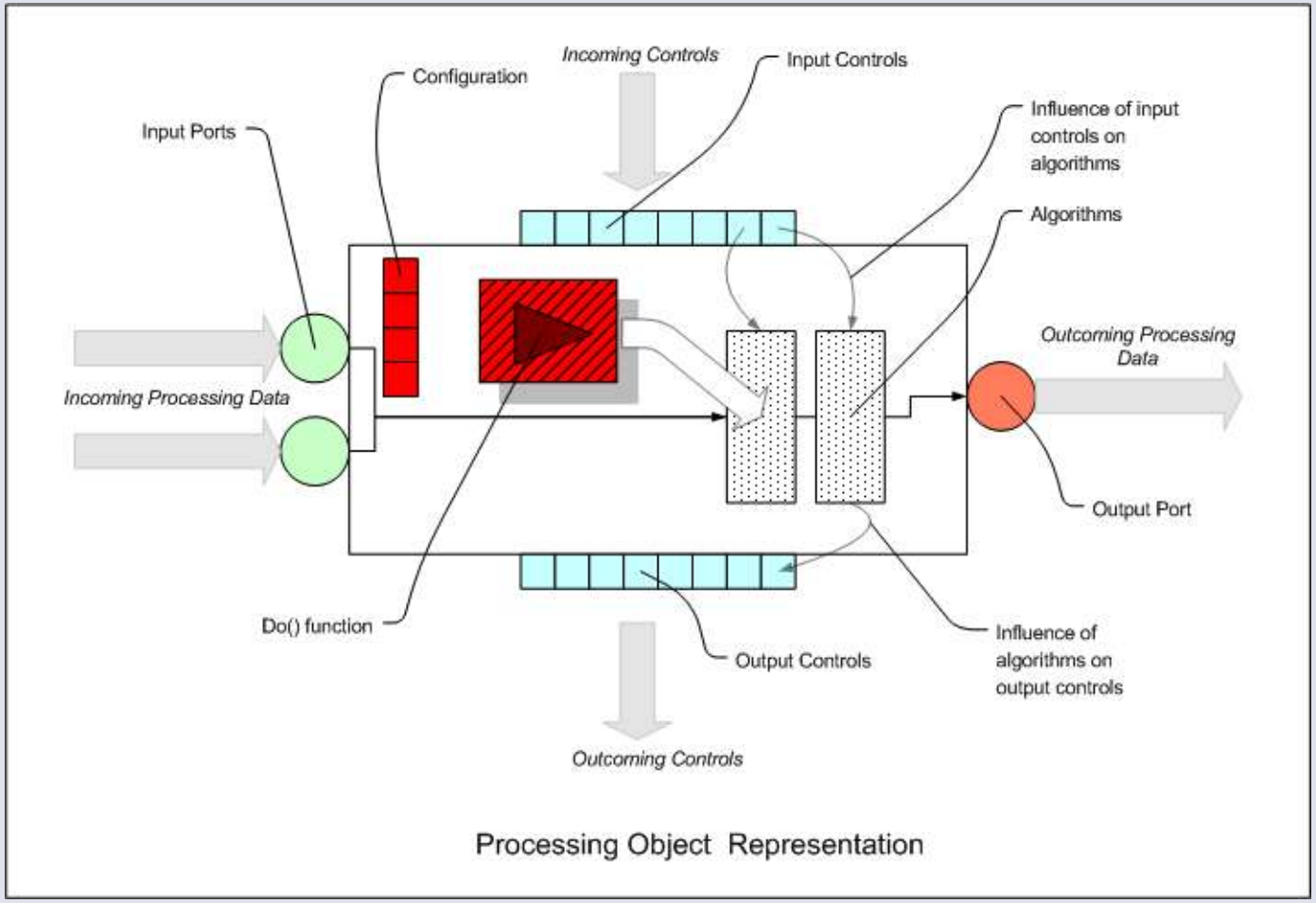
- Using a dynamic type

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

Processing Data

- All data involved in the process must be a subclass of the ProcessingData abstract class.
- Inputs/outputs to a Processing object must be Processing Data
- Processing Data are Dynamic Types
  - Most of its interface is automatically derived
- Any Processing Data has automatic XML persistency

Processing


Incoming Controls
Input Controls
Configuration
Input Ports
Influence of input controls on algorithms
Algorithms
Outcoming Processing Data
Incoming Processing Data
Output Port
Do() function
Output Controls
Influence of algorithms on output controls
Outcoming Controls
Processing Object Representation

Processing

- All the processing in CLAM has to happen inside a Processing class.
- The operation that triggers execution is the Do() operation, this is the only operation called from the external processing loop.
- Input and output from the processing can be done passing data to the Do() operation or using a more complex (better) Port mechanism.

Processing.Controls

- Control signals are treated differently
  - Controls generate “events” only when their value is modified.
  - Events travel to input controls located in another processing object that has previously been connected.
  - Processing objects can publish methods that act like functions called by input controls.
  - Processing objects can generate “events” for their output controls during the Do() execution.

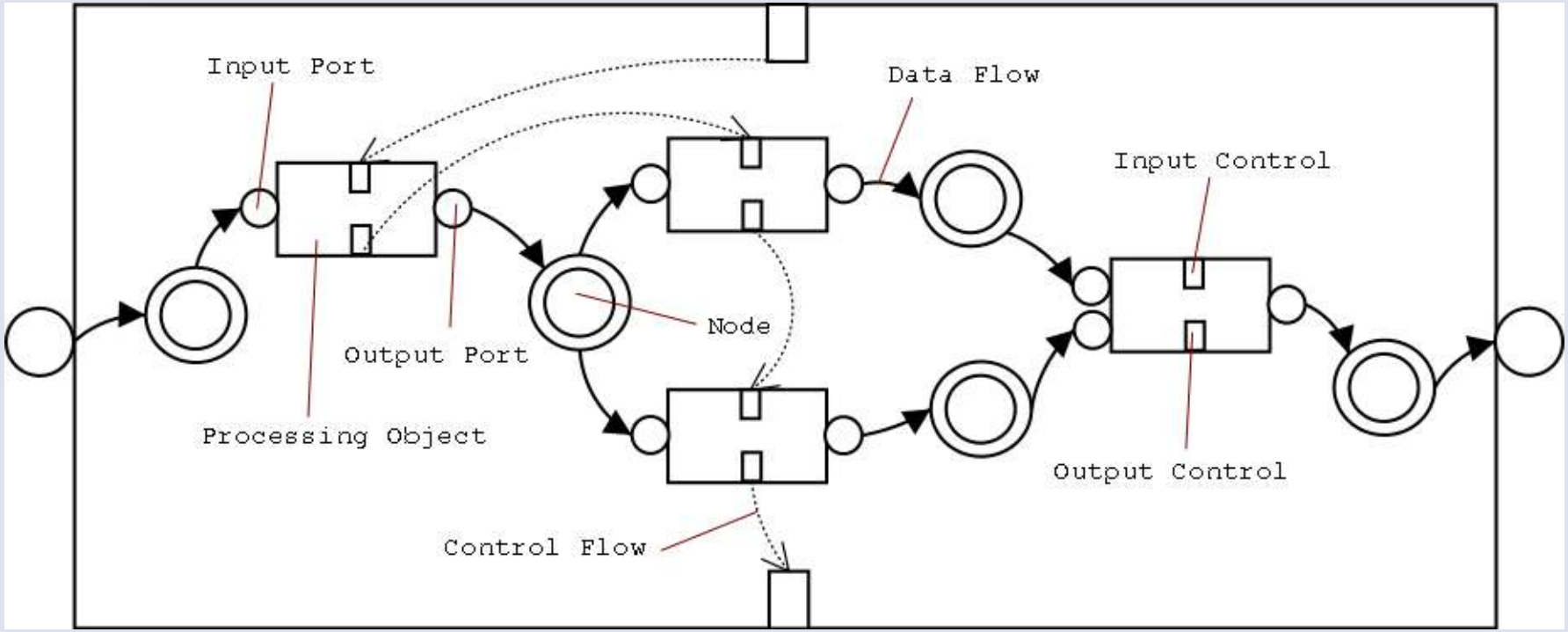
Processing.Configuration

- Processing classes have an associated configuration class.
  - Holds configuration parameters.
  - These parameters can also hold initial values for controls.
  - A configuration parameter can only be modified when the processing object is not in a "running" state.

CLAM Network model

- A CLAM network can be seen as a set of independent but connected Processing objects that encapsulate certain processes and collaborate for a common goal.
- The CLAM network is a graphical model of computation based on Dataflow Process Networks (very similar to Simulink or Ptolemy)
- Scheduling can be performed both statically and dynamically, depending on the particular application.

CLAM Network model



CLAM Network model

- CLAM makes a clear distinction between a synchronous data flow and an asynchronous control flow.
- Processing objects receive incoming data through their input ports and send processed data through their output ports (or in a similar way as arguments of the Do ()).

Processing Data Repository

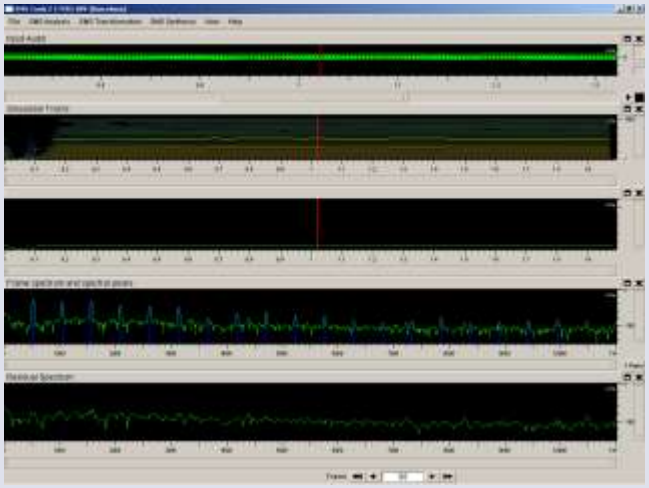
- Ready to use processing data:
  - Audio
  - Spectrum
  - SpectralPeakArray
  - Fundamental
  - Frame
  - Segment

Processing Repository

- Ready to use processing classes (almost 150 Processsing classes):
  - Analysis: FFT, spectral analysis, SMS analysis...
  - Arithmetic Operators
  - Input/Output Processings: Audio, AudioFile, MIDI, SDIF
  - Generators
  - Transformations
  - Synthesis

Available applications

- SMSTools

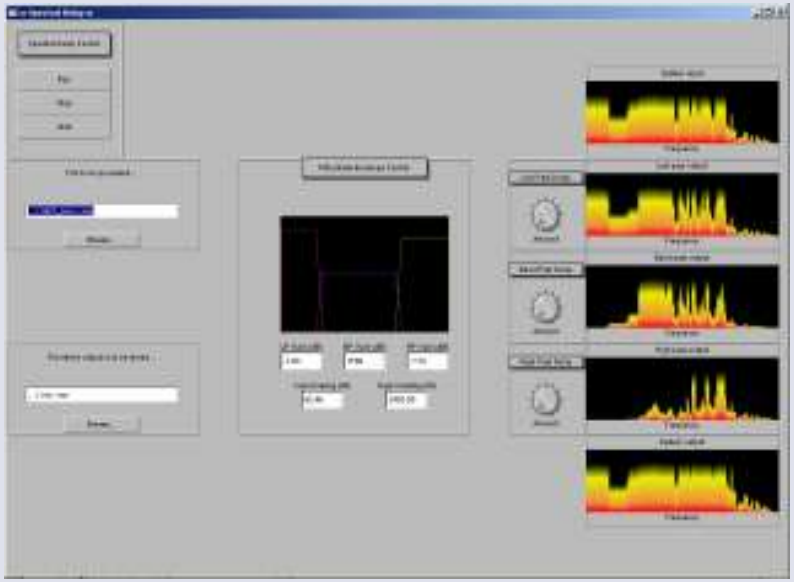


Available applications

- Salto

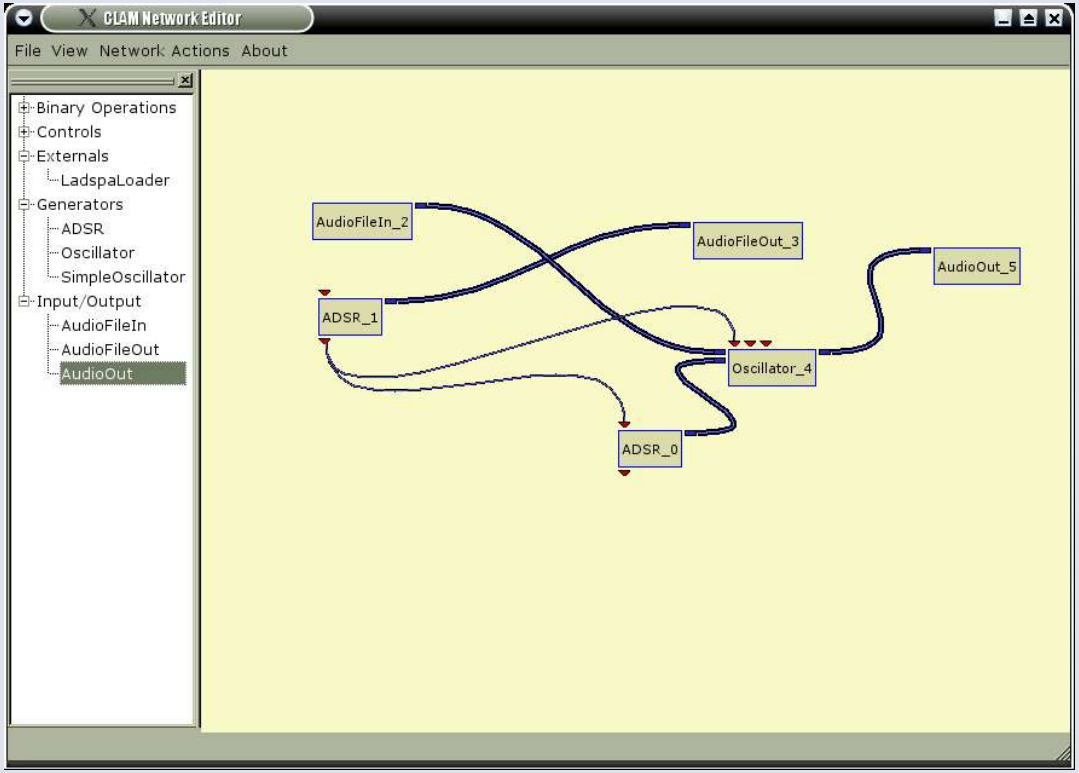
Available applications

- SpectralDelay



Available applications

- Network Editor



XML Interface

- Goal
  - Implementing data persistency
  - Ofering an easy way to store an object in XML format.
- Dynamic Types have an automatically derived XML (Store/Load) interface.
  - And therefore Processing Data.

XML Interface

```
-<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>
```

GUI

- CLAM offers its own infrastructure to integrate user interface into applications.
- It is made of a set of classes that implement an architecture derived from the MVC pattern and allows us to see data objects, processing objects and connexions in between them.
- Appart from that there are ready-to-use utilities:
  - Views of the most important Processing Data: Audio, Spectrum...
  - Debugging tools (Plots)

Audio I/O

- Using several libraries like Alsa, RtAudio or PortAudio CLAM offers audio i/o platform abstraction and integration into the CLAM model.
- The main class in CLAM audio input/output is AudioManager:
  - It is in charge of all administrative tasks related to the creation and initialization of audio streams using the AudioDevice class (which is system dependant).
- The first thing to do in order to use audio is to create an instance of the AudioManager class (singleton) that will be used by the rest of the audio I/O objects.

Audio I/O

- Then you can use the AudioIn and AudioOut classes in order to read or write Audio from your sound card.
  - These objectes are created using an AudioIOConfig object that specifies the device, the channel and the sampling rate.
  - These classes process mono channels (you have to instantiate one for each channel you want to stream).
- To specify the device you must use a string with the following syntax:

```
"ARCHITECTURE:DEVICE"
```

Audio I/O

- At this moment we have implemented the alsa and directx architectures (the latter using PortAudio, RtAudio or DirectX)
- Available devices depen on the hardware and system configuration (You may use the AudioDeviceList class in order to obtain a list of available devices).
- But if you don't specify the device or use the “default:default” string, AudioManager will automatically choose whatever device it thinks more appropiate for your system.

Audio I/O

- You can specify the channel you want for every AudioIn or AudioOut. Audio Manager will use this information for initializing internal management issues. We usually recommend 0 for L channel and 1 for R channel.
- 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: files

- We have implemented our own library for managing input/output of audio files.
- At the time being we only support raw, aiff and wav formats (a student is currently working on enhancing these).
- But what makes it different from most of the existing libraries is that it allows simultaneous reading/writing into the same file.

MIDI I/O

- MIDI I/O has been implemented using the PortMIDI library.
- The infrastructure is very similar to the Audio I/O one. We also have a MIDIManager.
- There is a MIDIIn class and a derived MIDIInControl that can be used to convert MIDI messages into CLAM controls.

MIDI Input

- The MIDIInConfig class has 3 parameters that specify what MIDI messages will be filtered to a particular MIDIIn object:
  - ChannelMask (bitmask)

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

  - MessageMask (bitmask)

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

  - Filter (filter to apply accotding to second bit in MIDI message)
- A MIDI file is treated as a MIDI device

Tools used in CLAM

- Programming language: C++
  - Flexibility
  - Efficiency
  - Standard vs. proprietaty language
- Programming tools
  - Windows: Visual C++ 7.X
  - Linux: gcc and other gnu tools
  - Mac OSX: gcc

Tools in CLAM

- CVS: code versioning control system for collaborative work (LinCVS recommended graphical front-end)
- Mantis: bug managing system based on a web interface
- Doxyen: generates html documentation from the javadocs comments inserted in the source files
- Mailing lists: clam@iua.upf.es

External libraries

- FFTW (FFT)
- Xercesc (XML parser that uses the DOM API)
- FLTK (GUI toolkit)
- Qt (GUI toolkit, not necessary but used in some applications)
- PTHREADS (multithreading on Windows)
- RtAudio, PortAudio, DirectX (for Windows audio)
- CppUnit (testing framework, only used for development)
- libsndfile: a library for reading and writing several audio file formats.
- Underbit's libmad: Mpeg Audio Decoding library.
- Xiph.org Ogg/Vorbis SDK: free implementation of Vorbis I encoder and decoder.
- id3lib: a library for parsing ID3 tags found on Mpeg audio bitstreams.

Conclusions

- Although there are still things to do, CLAM is already a usable framework that can yield interesting, efficient and robust applications.