

Building Large-scale Interactive Systems with OSC, Siren, CSL, and CRAM

Stephen Travis Pope

Center for Research in Electronic Art Technology
(CREATE)

Graduate Program in Media Arts and
Technologies (MAT)

University of California, Santa Barbara (UCSB)

stp@{create,mat}.ucsb.edu

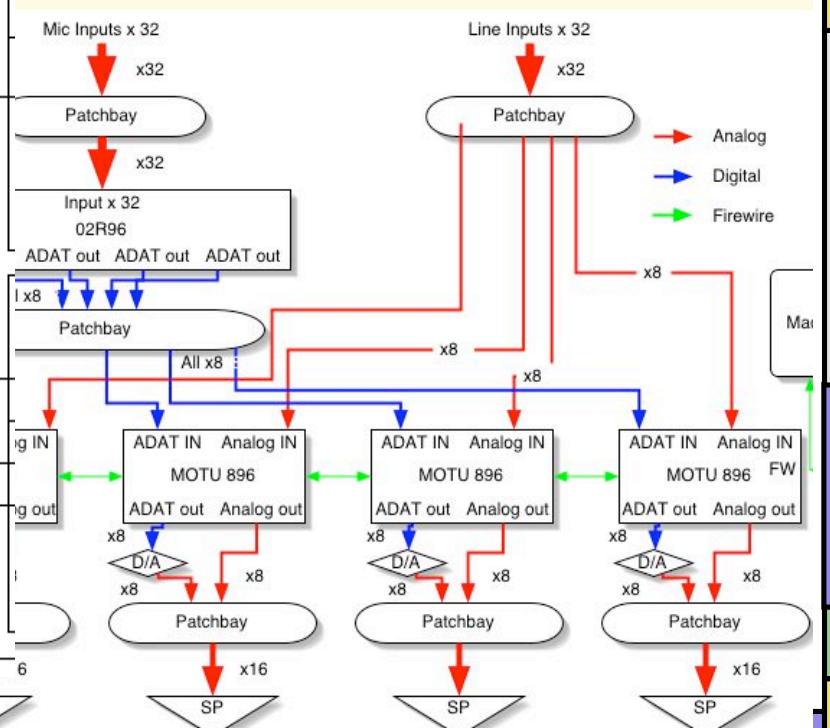
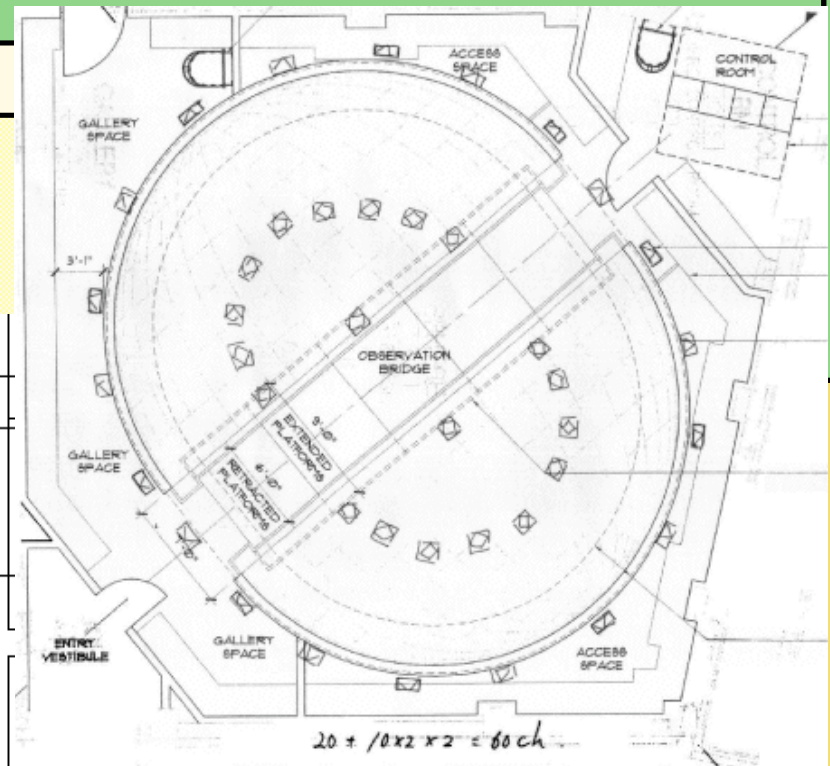
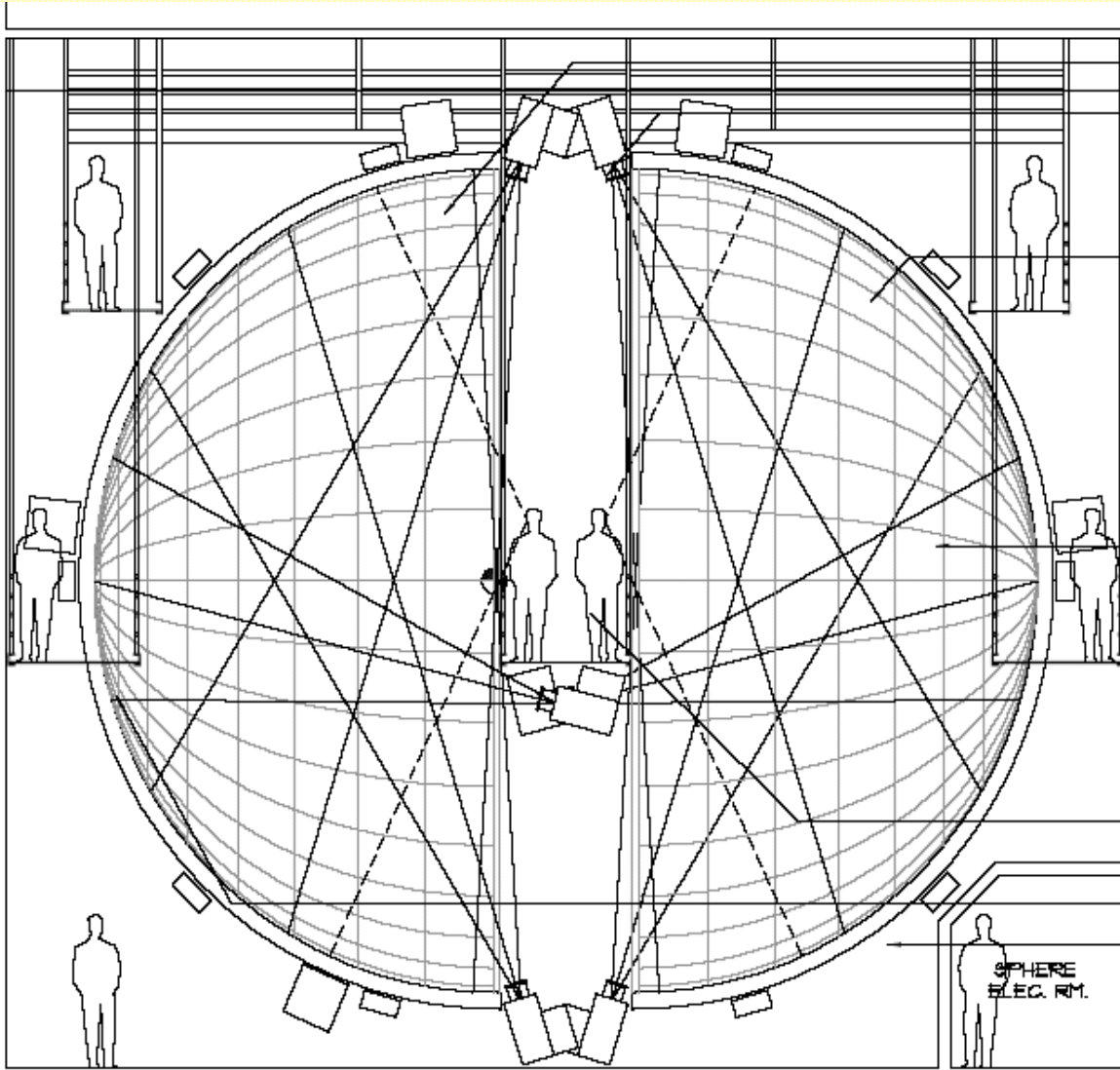
HW / SW Components

- **Siren:** Hierarchical / procedural representation for composers (OSC out)
- **CSL:** Scalable DSP framework (OSC srv)
- **CRAM:** Cluster management for distributed RT OO software (Mgr)
- **CNSI Sphere:** A really cool loud / bright / sensing space to play in!

Cal. NanoSystems Inst. @ UCSB

- MAT in CNSI: labs, studios, workshops, sphere
- CNSI compute infrastructure
 - Traditional vector supercomputer
 - 1024-node Linux cluster
 - Multimedia processing cluster (TBD)
- Sphere: 3-story I/O space
 - 12-channel overlapping video output
 - 128-channel sound output
 - Camera / microphone / sensor multi-modal input

CNSI Sphere



How? DSCP!

Distributed Sensing, Computation,
and Projection = MVC on steroids

Back-end application models are
scientific / numerical / simulation

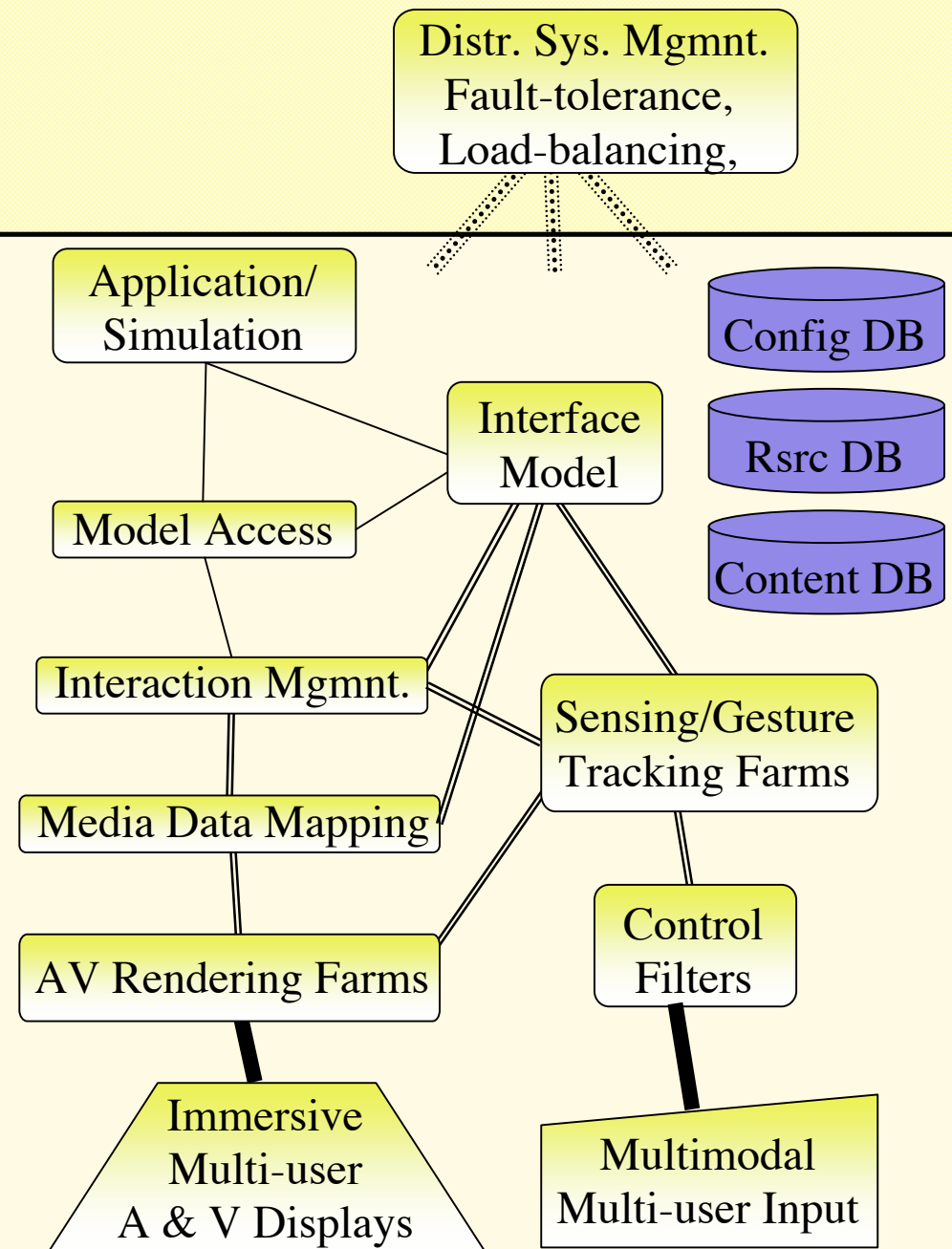
Multimodal multiuser **sensing/control**
and tracking / mapping farms

Application = sensing / tracking
policies + output data mappings

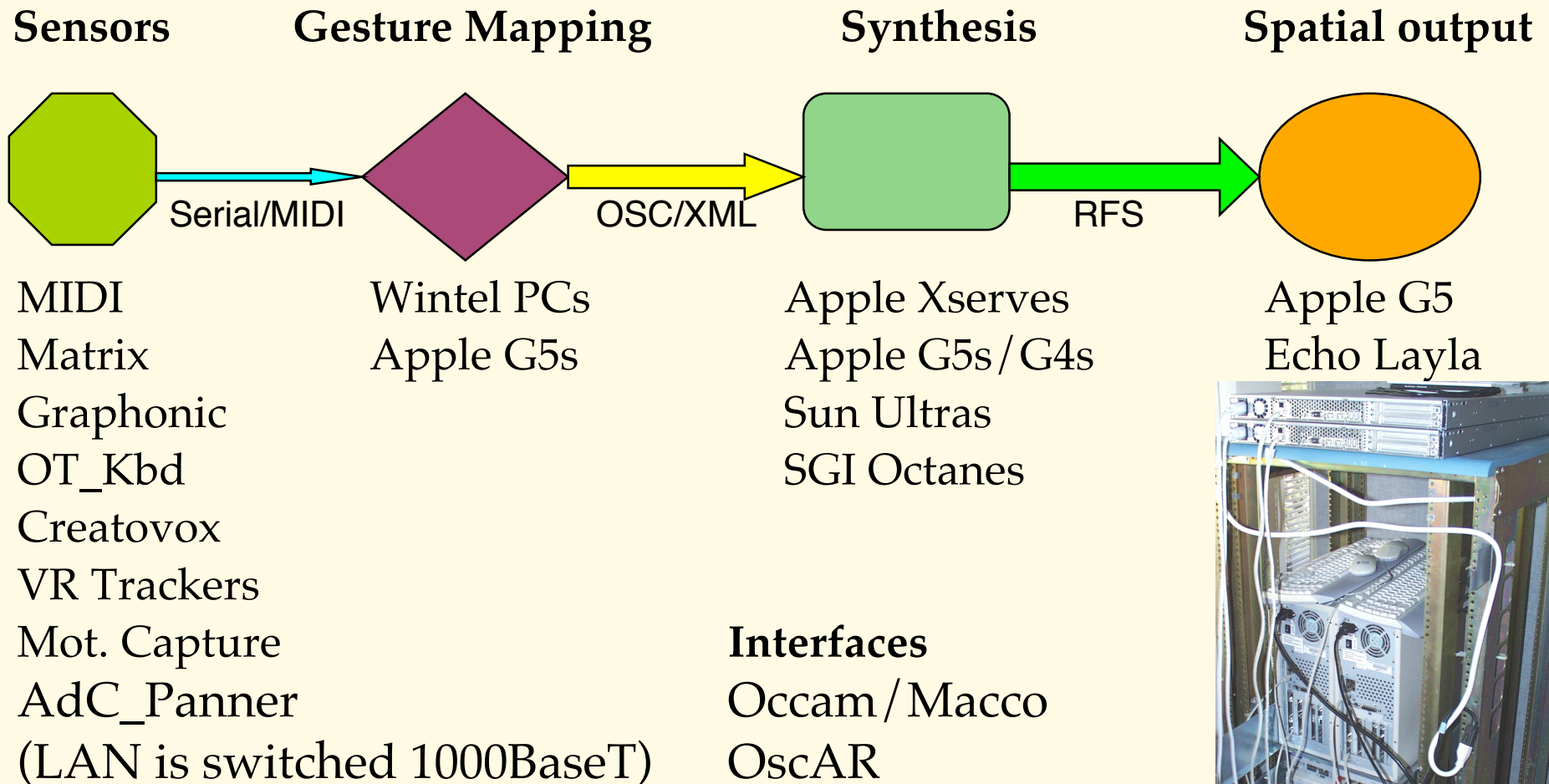
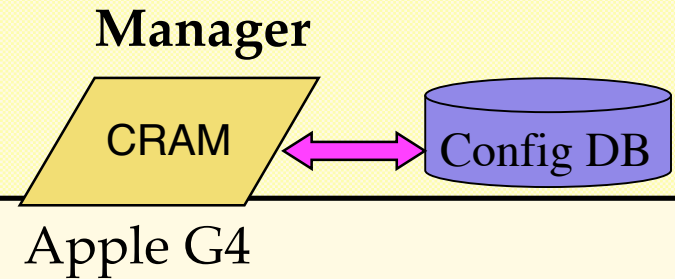
Presentation/interaction via CNSI
Sphere, LAN/WAN streaming

Infrastructure uses CRAM mgmnt

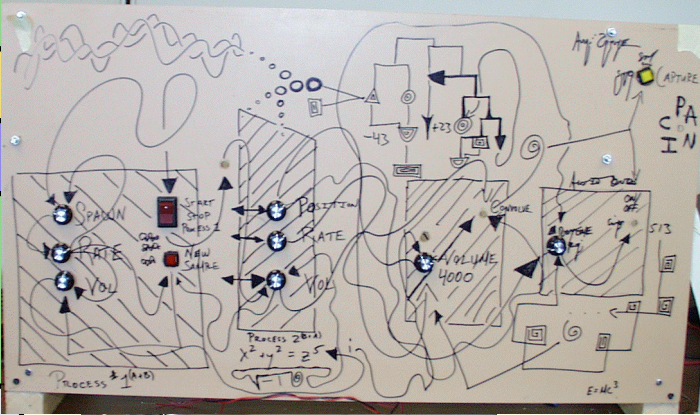
DBs for configurations, resources, and
media content (renderers)



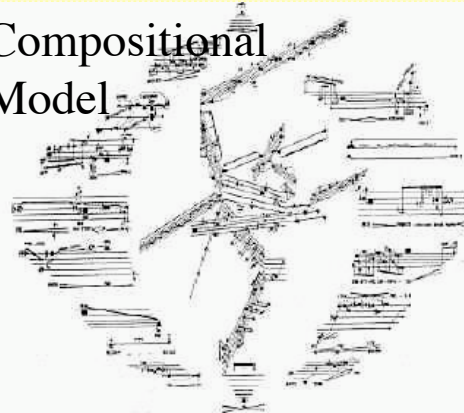
Current *Sphere-lite*



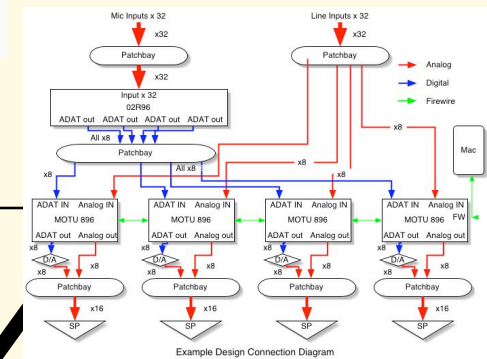
In Pictures



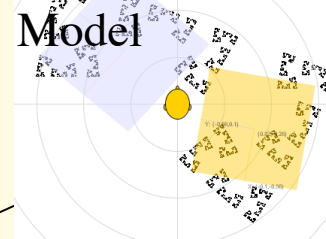
Compositional Model



Output Drivers



Spatialization Model

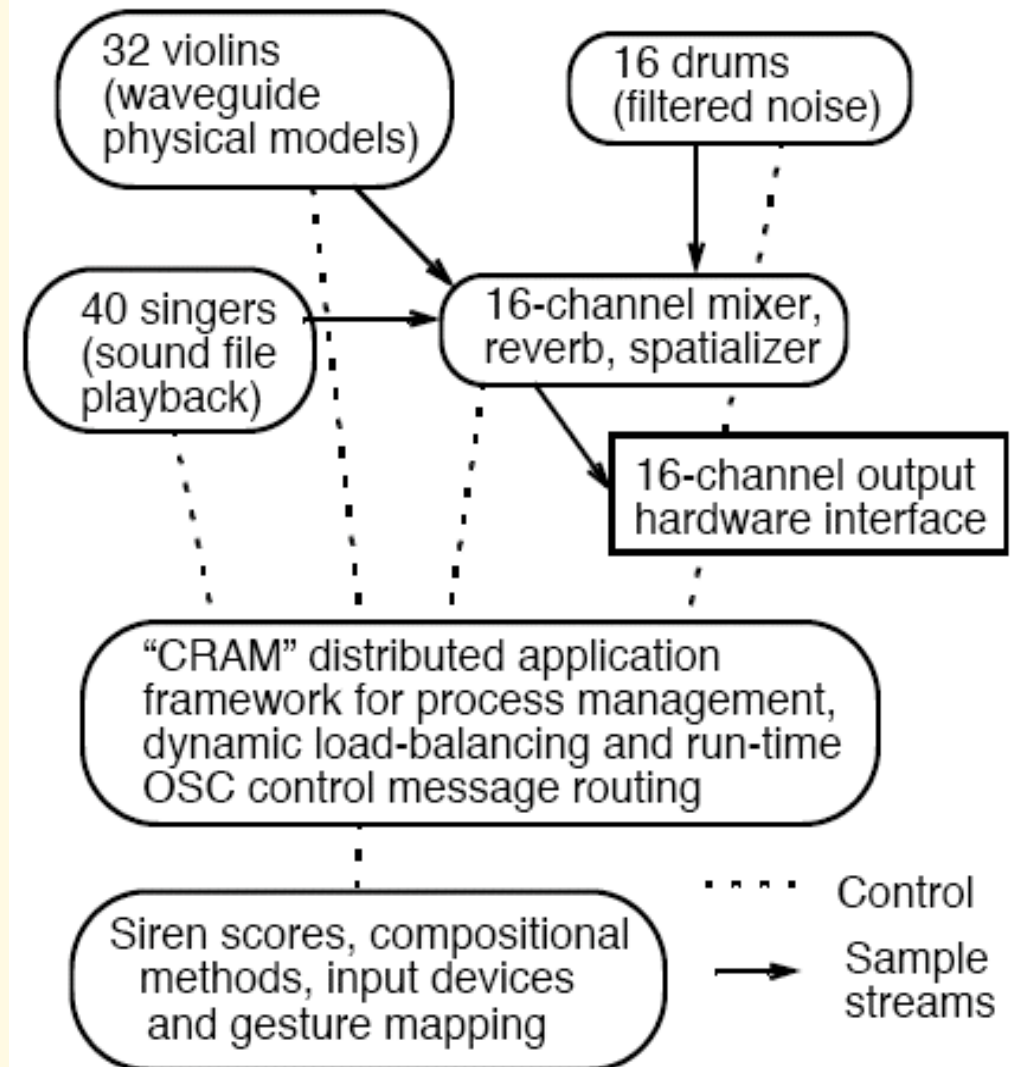


Gesture Sensors



Networked Synthesis / Performance

- Managed “orchestra-scale” sound synthesis, multi-modal gestural sensing and control, and pluriphonic projection (up to 128 channel output in the CNSI sphere)



Siren 2003 (VisualWorks)

Demo

The screenshot displays the VisualWorks NonCommercial Siren_7.4 environment. It features several windows:

- Function Editor:** A window showing a graph with multiple colored lines (blue, green, red, cyan) plotted on a grid, representing data or signal processing.
- Browser:** A window showing a class hierarchy. The selected class is `EventList`, which inherits from `MusicEvent`. Other classes visible include `AbstractEvent`, `ActionEvent`, `DurationEvent`, and `MusicEvent`.
- Source:** A window showing the source code for the `sentenceExample` class. The code includes a comment "Create an event list for a beautiful sentence.", a class definition `"EventList sentenceExample"`, and a method `EventList named: 'phrase2'` that generates a list of values for various parameters like duration, loudness, and phoneme.
- Display List View:** A window showing a musical score with notes and stems on a staff, representing the output of the `sentenceExample` class.

At the bottom, a **Source** window shows the definition of the `AbstractEvent` class:

```
Smalltalk.Siren defineClass: #AbstractEvent
superclass: #({Core.Object})
indexedType: #none
private: false
```

CSL “Hello world” Program

Demo

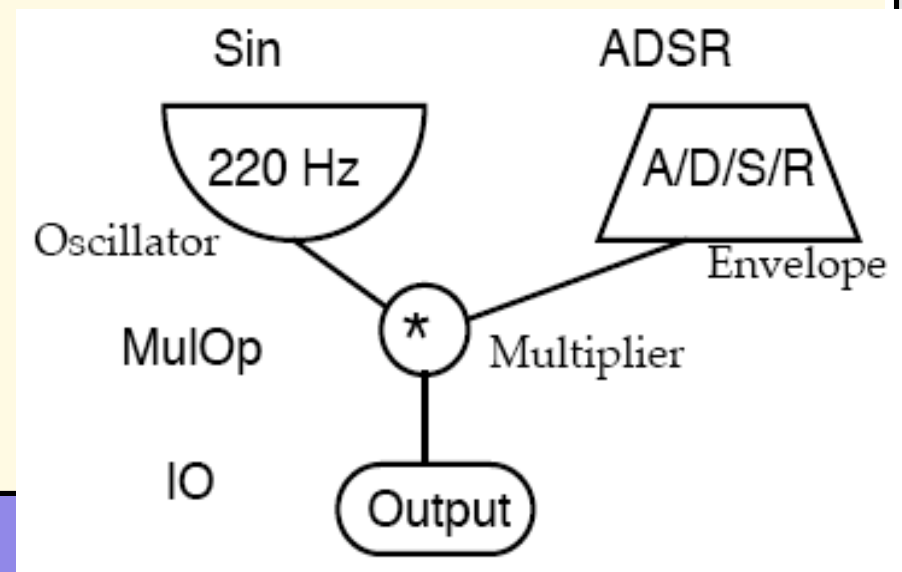
Sine wave with envelope

```
// Create a sine oscillator -- this is a comment
    Sine osc(220.0);

// Create an ADSR envelope -- args are (dur, a, d, s, r)
    ADSR env(3.0, 0.06, 0.2, 0.2, 1.5);

// Create a multiplier
    MulOp mul(osc, env);

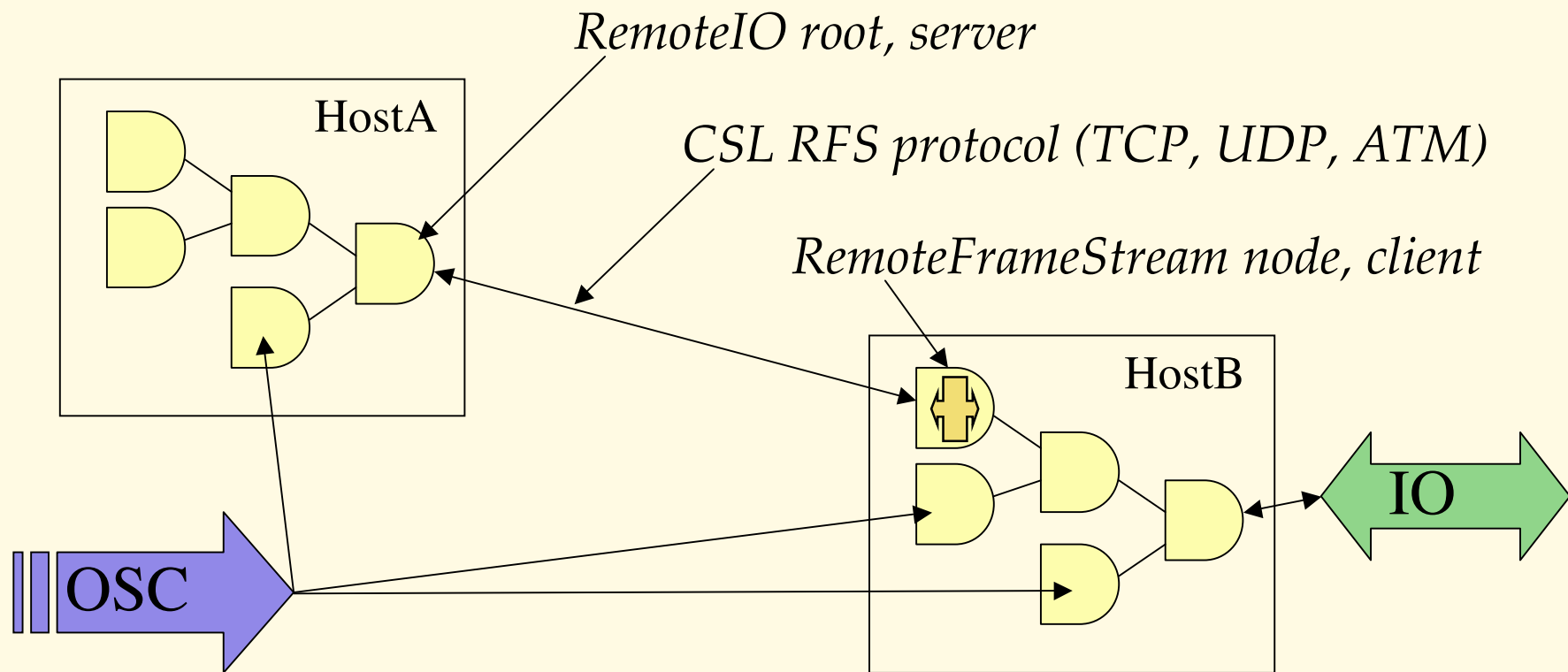
// Plug it into the output driver
    globalIO.set_root(mul);
```



Multi-host CSL Graphs

Demo

- Distributed sub-graph processing with RemoteIO and RemoteFrameStream, RFS protocol, buffering



CRAM Manager

- Network / Node
- Node / Service
- Application / Service
- Log / Control pane
 - Run-time monitor
 - Planning
 - DB play-back

The screenshot displays the CRAM System Manager interface. At the top, there are buttons for 'Refresh', 'Filter', and 'Exit'. The main area shows a tree view of the system hierarchy:

- CRAM System Manager
 - Network 'ridl' with 10 nodes
 - Node nomad is on. [X86/Linux] C: 26% M: 56% 0 Servi
 - Node fire is on. [PowerPC/MacOSX] C: 65% M: 68% 0 Servi
 - Node jerk is on. [SPARC/Solaris] C: 30% M: 60% 0 Servi
 - Node belly is on. [SPARC/Solaris] C: 40% M: 44% 0 Servi
 - Node stp is on. [PowerPC/MacOSX] C: 70% M: 45% 2 Servi**
 - Service 'clock1' is on.
 - Service 'clock2' is on.
 - Network 'create_lab' with 5 nodes
 - Application 'TestClocks' with 6 services
 - Application 'SirenCSL' with 4 services
 - Service CSLServer on node jerk
 - Service CSLServer on node belly
 - Service SirenGUI on node stp
 - Service CSL_Out on node fire
 - Application 'CSL_Farm' with 9 services
 - Application 'DRIVE' with 4 services
 - Application 'SuperColliderFarm' with 5 services

Below the tree is a 'Node Control' panel with buttons: Ping, Test, Log Options, Watch Options, Connect, List Services, Watch Node, Node Test, and Emergency Options. To the right of these buttons is a configuration table:

log_level	3
use_select	true
log_cache_size	8192
log_to_stdout	true
log_to_file	true
log_to_socket	false

At the bottom is a log pane showing system messages:

```
Mon Sep 15 11:02:05 2003: Info: [Get log tail]
Mon Sep 15 11:02:05 2003: Info: [Get log tail]
Mon Sep 15 11:02:16 2003: Info: [Ping service]
Mon Sep 15 11:02:20 2003: Info: [Ping service]
Mon Sep 15 11:05:54 2003: Info: [Get log tail]
```

A yellow arrow labeled 'Demo' points to the right side of the interface.

GestureSensor Drivers & Servers

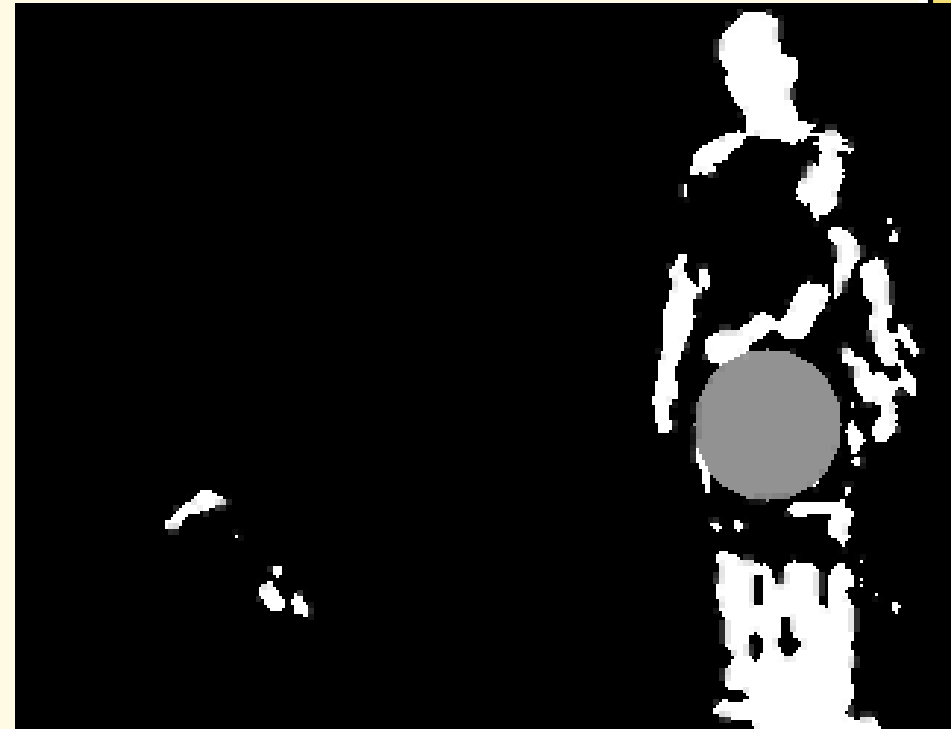
- Reusable sensor driver framework
 - Serial in, cacheing / differencing / throttling, OSC out
- GestureSensors: receive OSC or MIDI

```
void * mData;           // data array (typically a float *)
char * mCmd;           // OSC command (without the '/')
char * mTypeString;    // OSC type string, e.g., "ffff"
```

 - Event input thread mgmnt
 - Parsing and differencing
 - Map to static or global data or messages
- Subclasses
 - Glove, Ebeam, Matrix, FOBirds, AdC_Panner, etc.

CV-to-OSC

- Multiple-camera 3D motion tracking of multiple sources
- Data mapping for sound synthesis and transformation algorithms
- Intelligent trans-media system that learns and adapts, based on memory of the actions and states of the sensor space



Siren (MODE, HSTK, DoubleTalk)

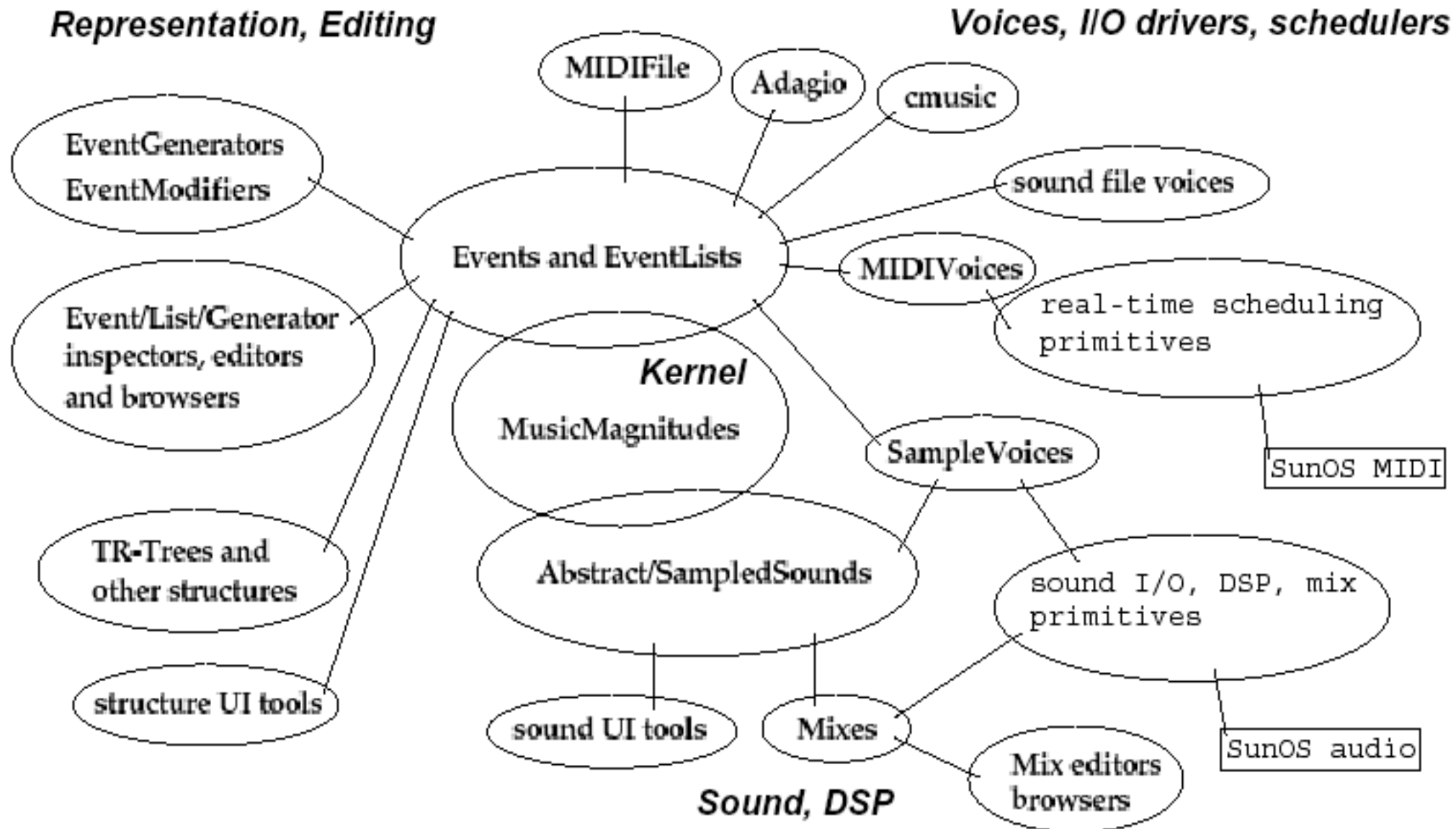
- Smalltalk-based object-oriented framework for sound / music description and processing, under development since 1984
- Focus on structure representation, control mapping, and composition, rather than on performance, DSP, or notation
- API / Platform for music representation and composition language development

What's Siren?

[440 Hz, (1/4 beat), 44 dB]
evtList mapPitches: gamut.
evtList playOn: Voice default.

- **Smoke** music representation language
 - Music magnitudes, events, event lists, generators, modifiers, struct. algorithms, ...
 - Organize timing, tuning, timbre, space, gesture, grouping, versioning
- **I/O voices** (players, property-parameter mappers) for many formats: (m11-SC3) note lists, OSC, MIDI, XML, CORBA, ...
- **Multi-threaded RT scheduler**
- **GUI widgets** and apps for music
- **(OO/R)DBMS** interfaces for **persistence**

Siren Components (1992)



Siren 2003 (VisualWorks)

Demo

The screenshot displays the VisualWorks NonCommercial Siren_7.4 environment. It features several windows:

- Function Editor:** A window showing a graph with multiple colored lines (red, green, blue, cyan) plotted on a grid, representing data or signal processing.
- Class Browser:** A window showing a hierarchy of classes. The **EventList** class is selected, showing its subclasses: `AbstractEvent`, `ActionEvent`, `DurationEvent`, and `MusicEvent`.
- Source Code:** A window showing the source code for the `sentenceExample` class. The code includes a comment "Create an event list for a beautiful sentence.", a class definition for `sentenceExample`, and a method that generates an `EventList` named 'phrase2' with specific parameters and values.
- Display List View:** A window showing a musical score visualization. The score is displayed on a staff with notes and stems, and is labeled with pitch classes `c6`, `c7`, `c8`, and `c9`.

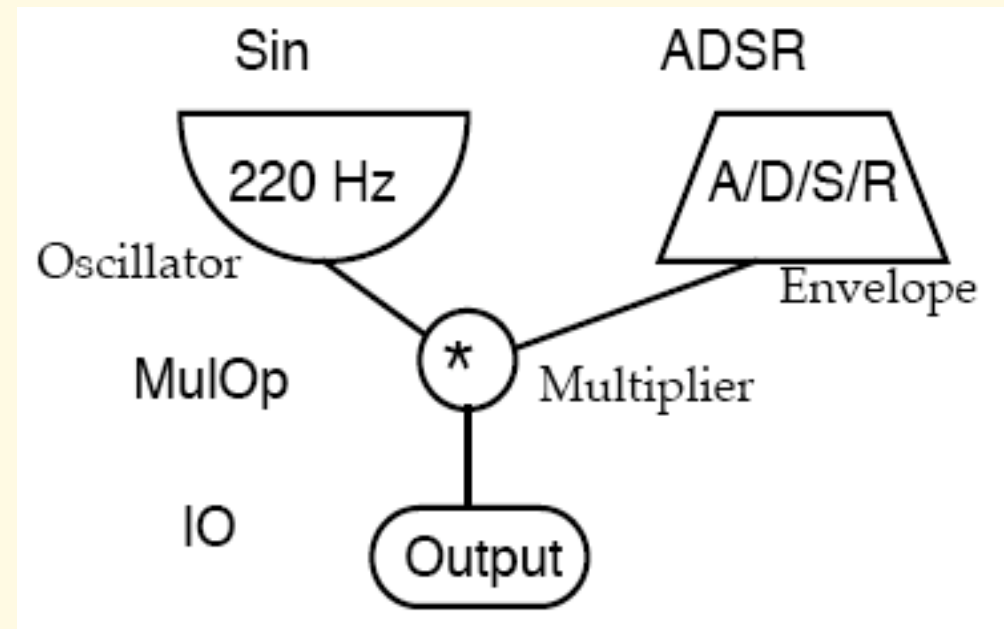
The bottom of the interface shows a code editor with the following code snippet:

```
Smalltalk.Siren defineClass: #AbstractEvent
superclass: #({Core.Object})
indexedType: #none
private: false
```


The CREATE Signal Library (CSL, “sizzle”) (“chill?”)

Demo

- General-purpose, portable C++ framework for distributed, real-time digital audio synthesis and processing
- Used for stand-alone applications, plug-ins, OSC servers, etc.



CSL Relatives

- Like Cmix, STK, Siren, JSyn, MxV, or CLM
 - Delivered as a library in a general-purpose programming language
- Unlike SuperCollider, Csound, Max
 - Not its own language
 - No scheduler
 - Uses C++ development environment

CSL3 Basics

- **Buffer** objects (1-4 classes)
 - Multichannel non-interleaved sample storage
 - “Smart” object, not just a (float **), ptr. mgmnt.
 - Handle malloc/free, filling statistics, etc.
- **FrameStream** classes (Ugens) (many)
 - Respond to the message `next_buffer(input, output)`
 - Processors have a FrameStream as input
- **Mix-in** classes (vs. wrapper classes)
 - Phased, Positionable, Writeable, Cacheable, etc.

“Hello world” in CSL

Demo

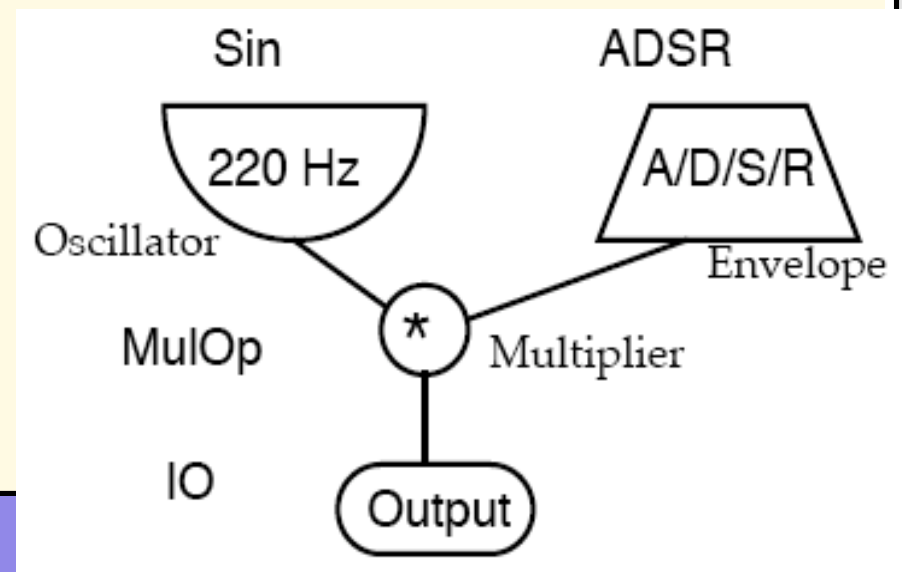
Sine wave with envelope

```
// Create a sine oscillator -- this is a comment
    Sine osc(220.0);

// Create an ADSR envelope -- args are (dur, a, d, s, r)
    ADSR env(3.0, 0.06, 0.2, 0.2, 1.5);

// Create a multiplier
    MulOp mul(osc, env);

// Plug it into the output driver
    globalIO.set_root(mul);
```



CSL Sources, Controls, and Processors

- **Sources**

- Oscillators (perfect, BL), SumOfSines, Noise, SoundFiles, Chaotic/IteratedFS, IFFT, Physical Models, Granulators, Signal windows

- **Control**

- Envelopes, LFOs, LFNoise, ProbDists, DynamicVariables, OSC, MIDI, GUI, CORBA, XML, note lists, Feature extractors, Input followers

- **Processors**

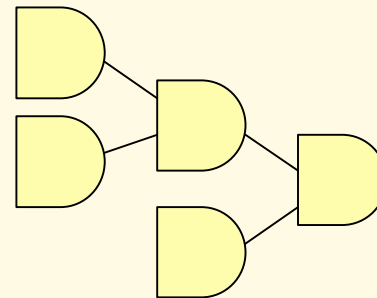
- Operators, Mixers, Filters/banks, Reverbs, (N-M)Panners, DelayLines, FDN, WaveShape, Lo-latency Convolution, FFT/IFFT, LPC/FIR

- **Support**

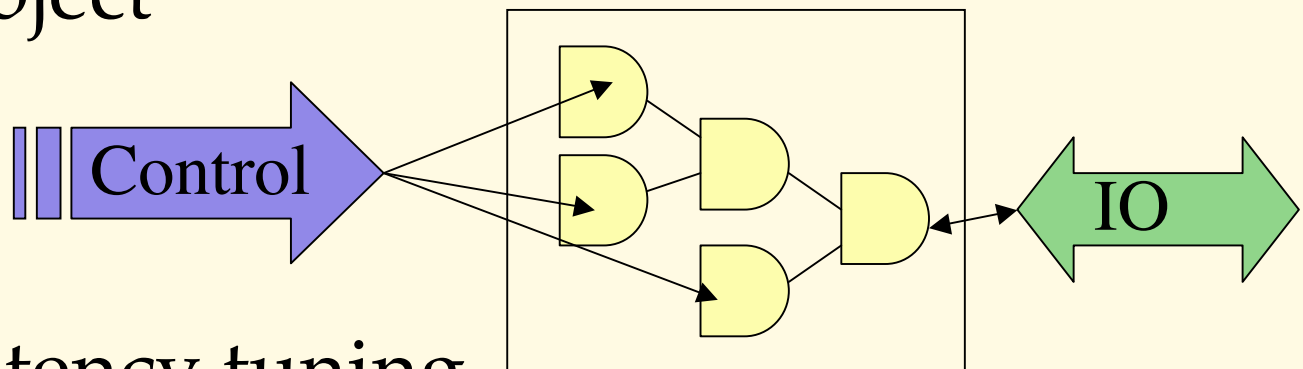
- RingBuffer, ThreadedFrameStream, BlockResizer, RateConvertor, Splitter/Joiner, FanOut (needed), Interleaver/Deint., Test main(s)
- Tools: FIR/Reverb IR Design, Spectrum DBs, Control-mapping

The Big Picture of CSL

- Basic DSP graph



- Connected to control input (OSC, MIDI, GUI, CORBA, XML), and IO object



- Buffering and latency tuning

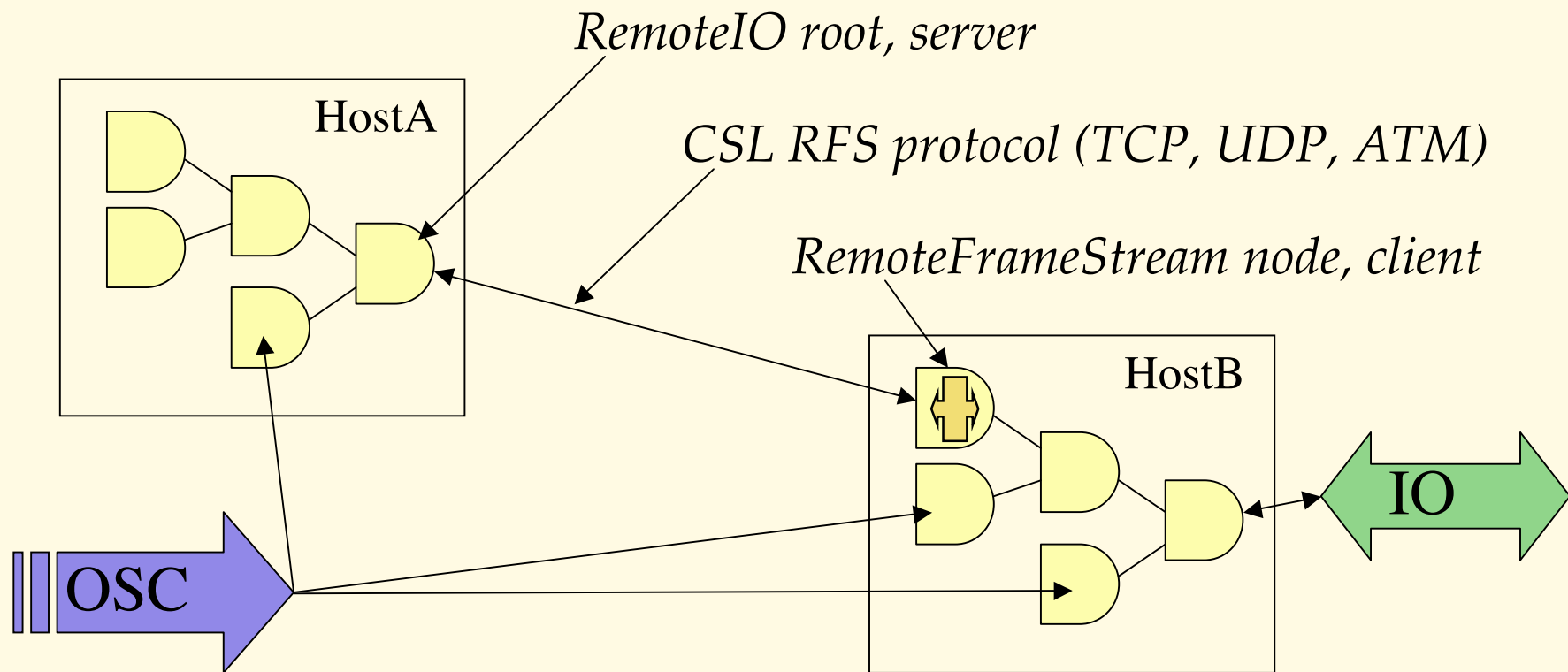
CSL DSP Graph Flexibility

- Sub-graphs can run at different:
 - Sample rates (for control),
 - Buffer sizes (for transforms),
 - Numbers of channels (for efficiency),
 - Buffer formats (interleaved or not),
 - In different threads, etc.
- These can be changed (within reason) at run-time (e.g., for load- or traffic-balancing)

Multi-host CSL Graphs

Demo

- Distributed sub-graph processing with RemoteIO and RemoteFrameStream, RFS protocol, buffering



Instruments and OSC / MIDI / XML

- Instrument object

- Holds onto a DSP graph; adds “reflective” accessors
- Generates OSC address spaces, MIDI maps, etc.
- Server main() function loads an instrument library and publishes an address space on a listener socket

- Example:

// C++ accessor decl.

```
list[0] = new Accessor("du", set_duration_f, CSL_FLOAT_TYPE);
```

```
list[1] = new Accessor("am", set_amplitude_f, CSL_FLOAT_TYPE);
```

// Produces:

```
/i1/
```

instrument 1's OSC address space

```
/i1/du:
```

set-duration command

```
/i1/am:
```

set-amplitude command

GestureSensor Drivers & Servers

- Reusable sensor driver framework
 - Serial in, cacheing / differencing / throttling, OSC out
- GestureSensors: receive OSC or MIDI

```
void * mData;           // data array (typically a float *)
char * mCmd;           // OSC command (without the '/')
char * mTypeString;    // OSC type string, e.g., "ffff"
```

 - Event input thread mgmnt
 - Parsing and differencing
 - Map to static or global data or messages
- Subclasses
 - Glove, Ebeam, Matrix, FOBirds, AdC_Panner, etc.

CSL main() for OSC Processing

```
// Set up OSC address space root
init_OSC_addr_space();

// EITHER: add the instrument library OSC addr. space
setup_OSC_instr_library(library, numInstruments);

// OR: create a background thread for a GestureSensor
Thread * aThread = ThreadPthread::MakeThread();
aThread->fork_thread(GS_thread_fcn, & someArgument);

// start the I/O callback thread
GlobalIO->start();

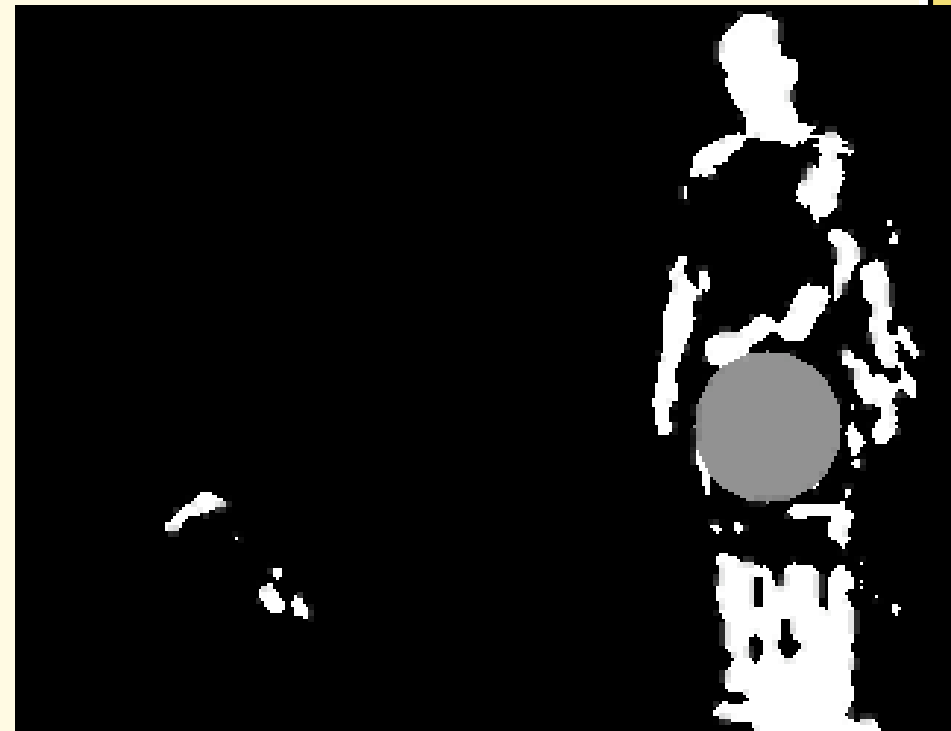
// Run the OSC I/O loop function (doesn't return)
main_OSC_loop(theUDPPort);
```

OSC with a Shell Script

```
# Shell script to test sending OSC messages to CSL
# Create a convenient alias
alias ssoo "sendOSC -h localhost 54321"
# Play a note on instrument 1
ssoo /i1/p;          sleep 3
# Set a value and play another note
ssoo /i2/cf,50.0;   ssoo /i2/p;          sleep 3
# play a note with parameters: dur/amp/car/mod/ind
ssoo /i4/pn,4.0,0.3,220.0,357.4,3.0;  sleep 4
# load a sound file
ssoo /i8/fi,"$CSL_DATA/shine.snd"
# play a sampled sound
ssoo /i8/p;          sleep 1
```

CV Input to OSC

- Implement multiple camera 3D motion tracking of multiple sources.
- Construct an intelligent trans-media system that learns and adapts, based on memory of the actions and states of the sensor space.
- Map the data to sound synthesis and transformation algorithms that will provide evocative and meaningful results.



Managing Siren and CSL: CRAM

- CRAM: Yet another **Distributed Processing Environment** (DPE, Cluster Mgmt. literature)
- Framework to deploy, start/stop, and monitor multi-host distributed real-time OO applications
- Provides fault-tolerance and load-balancing*
- CRAM is 3rd-gen. DPE implementation at CREATE (1996-2004) (HPDM/TAO, Yellow/CORBA_AV)
- Designed for robustness, simplicity, and low overhead; limited services and scalability/replication

CRAM Manager

- Network / Node
- Node / Service
- Application / Service
- Log / Control pane
 - Run-time monitor
 - Planning
 - DB play-back

The screenshot displays the CRAM System Manager interface. At the top, there are buttons for 'Refresh', 'Filter', and 'Exit'. The main area shows a tree view of the system hierarchy:

- CRAM System Manager
 - Network 'ridl' with 10 nodes
 - Node nomad is on. [X86/Linux] C: 26% M: 56% 0 Servi
 - Node fire is on. [PowerPC/MacOSX] C: 65% M: 68% 0 Servi
 - Node jerk is on. [SPARC/Solaris] C: 30% M: 60% 0 Servi
 - Node belly is on. [SPARC/Solaris] C: 40% M: 44% 0 Servi
 - Node stp is on. [PowerPC/MacOSX] C: 70% M: 45% 2 Servi**
 - Service 'clock1' is on.
 - Service 'clock2' is on.
 - Network 'create_lab' with 5 nodes
 - Application 'TestClocks' with 6 services
 - Application 'SirenCSL' with 4 services
 - Service CSLServer on node jerk
 - Service CSLServer on node belly
 - Service SirenGUI on node stp
 - Service CSL_Out on node fire
 - Application 'CSL_Farm' with 9 services
 - Application 'DRIVE' with 4 services
 - Application 'SuperColliderFarm' with 5 services

Below the tree is a 'Node Control' panel with buttons: Ping, Test, Log Options, Watch Options, Connect, List Services, Watch Node, Node Test, and Emergency Options. To the right of these buttons is a configuration table:

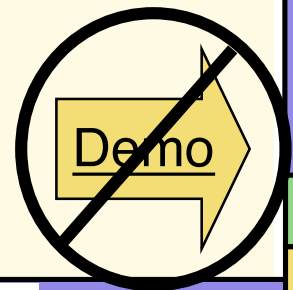
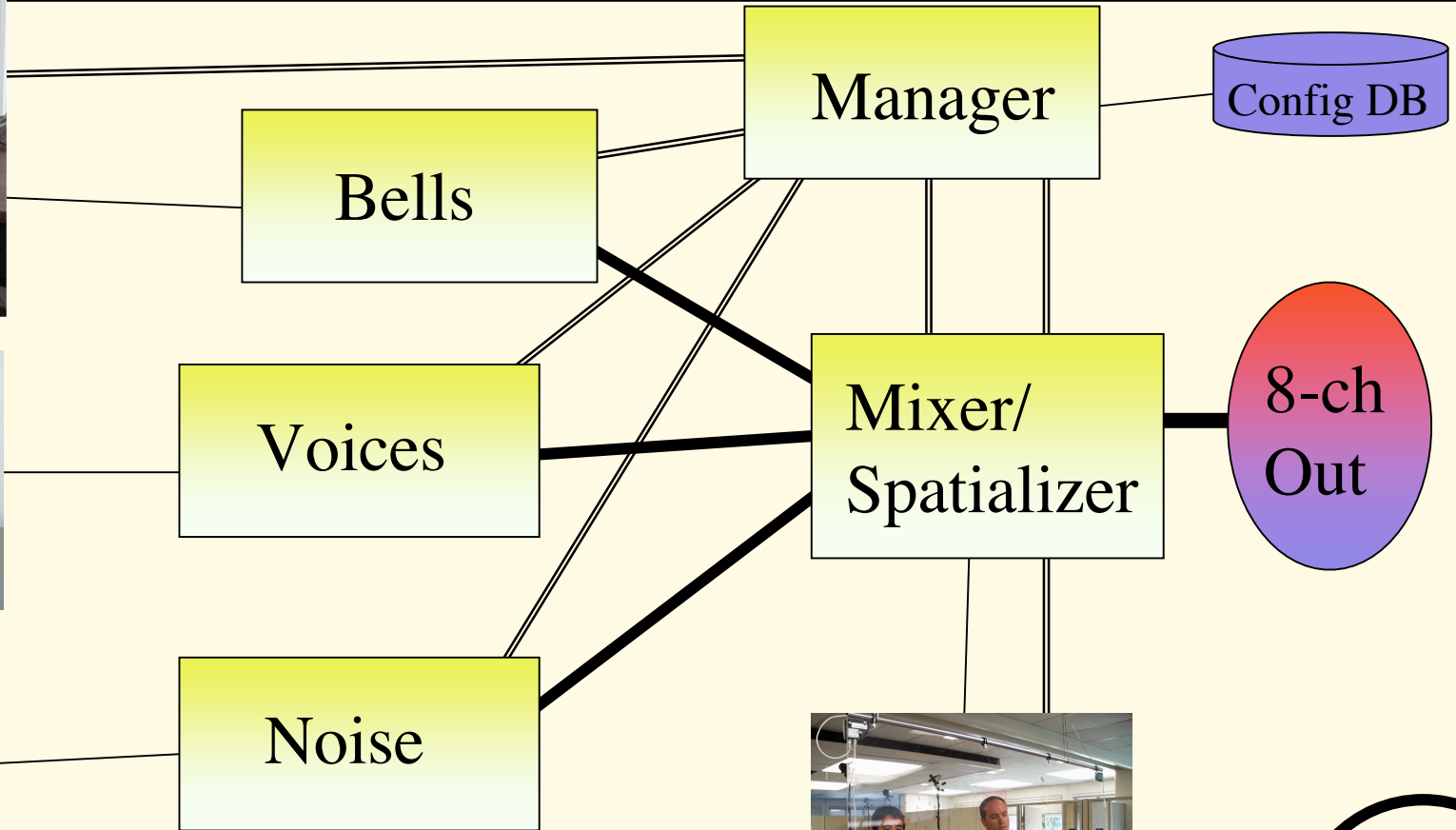
log_level	3
use_select	true
log_cache_size	8192
log_to_stdout	true
log_to_file	true
log_to_socket	false

At the bottom is a log pane showing system messages:

```
Mon Sep 15 11:02:05 2003: Info: [Get log tail]
Mon Sep 15 11:02:05 2003: Info: [Get log tail]
Mon Sep 15 11:02:16 2003: Info: [Ping service]
Mon Sep 15 11:02:20 2003: Info: [Ping service]
Mon Sep 15 11:05:54 2003: Info: [Get log tail]
```

A yellow arrow labeled 'Demo' points to the right side of the interface.

CRAM Configuration for CSL



Related Projects at CREATE

- Auralizer & VRML
- Pulsar Generator
- Creatovox
- MusicVisualization
- FMAK DB
- TimeMachine
- InteractEMGroup
- Creatophone
- Time-DDecomp
- SC_3 Work

