MuSA.RT and SAI
Data Stream Processing for Music and More

Alexandre R.J. François
afrancoi@usc.edu
http://iris.usc.edu/~afrancoi

Prelude
A system for real-time analysis and interactive visualization of tonal patterns in music

From Algorithms to System

- Tonal analysis
  - Spiral Array model
  - CEG algorithms
- Input devices
  - MIDI keyboard
  - Gamepad
- Rendering
  - 3D graphics (OpenGL)
  - MIDI synthesizer

System Requirements

- Efficiency
  - Real-time processing: analysis, rendering, etc.
- High throughput
  - Interactive rendering
- Low latency
  - Immediate (perceived) reactivity
- Synchronization
  - Graphics and sound

Roadmap

- SAI
  - Software Architecture
  - The SAI style
  - Architectural properties
  - Architectural Middleware
  - Example designs
- MuSA.RT
  - Background and context
  - System design
  - Algorithms and implementation
  - System performance

SAI
Software Architecture for Immersipresence
Introduction

SAI is a software architecture model for designing, analyzing and implementing applications performing distributed, asynchronous parallel processing of generic data streams.

Software Architecture

- The road from equations to software product
  - From algorithm to implementation-proof of concept
  - From proof of concept-test code to integrated systems
- Explicit system structure
  - Technical basis for design
  - Blue-prints for implementation
- Project management
  - Planning: cost estimation, resource allocation
  - Separation of concerns
- Question #1: what architectural style(s)?

SAI Goal and Scope

- Framework for the design, analysis and implementation of complex software systems
- Hard performance constraints (throughput and latency)
- Distributed asynchronous parallel processing of generic data streams

Key Concepts in SAI

- Time
  - All data can be considered a data stream in time
  - Perceptual constraints: precedence and simultaneity
  - Explicit attribute of all data
- Data classes
  - Volatile: stream samples
  - Persistent: process parameters, dynamic structures
  - Asynchronous parallelism

A Hybrid Model

\[ t_0 < t_1 < t_2 < t_3 \]

Message-passing

\[ a_0, b_0, c_0, d_0 \]

Processing

\[ a_3 \]

Persistent data

\[ a_2, b_2 \]

Volatile data

\[ a_1, b_1, c_1 \]

The SAI Style

- Components
  - Sources: system states (shared memory)
  - Cells: processing (no state)
- Connectors
  - Cell-cell: volatile data samples (stream)
    - At most one upstream cell
    - Any number of downstream cells
  - Cell-source: persistent data access
    - A cell is connected to exactly one source
    - A source is connected to any number of cells
- Topological constraints
  - Active (stream): process dependency
  - Passive: concurrent access

\( [\text{ISE2003}] \)
Data Model

- Pulses
  - Synchronization structure (time stamp, duration)
  - Active: volatile, flow down stream connections
  - Passive: persistent (dynamic), held in sources
- Pulse is organized as a hierarchy of nodes
  - Nodes are atomic data units
  - Encode state and methods
- Extensible set of nodes
  - Specialized into data structures
  - Encapsulate existing ones (APIs and libraries)

Underlying Processing Model

- A cell processes incoming active pulses in separate threads (asynchronous model)
- Active pulse
  - Cell process may read and add to an active pulse
  - Concurrent read access
  - Strict message passing communication
- Passive pulse
  - Cell process may read and modify a passive pulse
  - Concurrent shared memory access (read/write)

Notation for Architectural Designs

- Levels of detail provide different views
- Conceptual architecture
  - Data streams
  - Cell functionalities
- Logical architecture
  - Component interfaces: nodes, cells
- Execution architecture
  - Explicit cell processes and data structures

Architectural Properties

- Facilitates system design
  - Intuitive architectural style, preserve natural flow of data
  - Unified processing model and unified data model
  - Achievable optimality and scalability
- Facilitates distributed development, fast integration, code reusability
- Facilitates system maintenance, modification and evolution (change in algorithm and in function)
- Naturally support concurrent execution and distributed processing

Achievable Optimality

- Asynchronous parallel processing: decouple latency and throughput
- Optimal design (actual performance depends on implementation)

Scalability
Architectural Middleware: MFSM

- Open source project
  - mfsm.SourceForge.net
  - C++ (STL) / Windows
  - Multithreaded (assume fair scheduling)
- FSF Library: extensible set of classes that implement
  - All SAI style elements: source and cells* (connectors do not have an active role)
  - Related elements: nodes*, filters, handles, system, factories, node and cell specializations

MFSM Pluses

- Modules: specializations that implement specific algorithms or functionalities
  - Video, MIDI, sound, graphics
  - Encapsulate existing libraries: Windows API, OpenGL, OpenCV, etc.
- Documentation and tutorials
  - User guide
  - Reference guide
  - Tutorials (Image, Video, etc.)

Example Designs

- From real-time computer vision to integrated, distributed interactive systems...
  - Real-time video segmentation and tracking (IRIS)
  - Video Painting (IRIS)
  - Handheld mirror simulation (IMSC)
  - Live video in animated 3-D graphics (IMSC)
  - IMSC Communicator (IMSC)
  - Distributed collaborative game (CS 599 – IMS)

Real-Time Video Segmentation and Tracking

- Live input
- Live input frame
- Foreground detection
- Connected components
- Tracking
- Composite image
- Display (Composite image)

Video Painting

- Live input
- Input frame
- Image pyramid computation
- Image pyramid
- Transformation
- Mosaic generation
- Output
- Display (Mosaic)
Handheld Mirror Simulation

Live Video in Animated 3D Graphics

IMSC Communicator

Distributed Collaborative Game (class project)

Summary

MuSA.RT

- SAI: software architecture for distributed parallel processing of data streams
  - Subsumes pipes and filters and shared repositories
  - Interaction, efficiency, scalability
- MFSM: open source architectural middleware
  - C++ / Windows
  - Use with existing libraries: OpenCV, DirectX, etc.
- Applications: Integrated Multimedia Systems
  - Cross-Disciplinary
  - Interactive
**Introduction**

- Real-time analysis and interactive visualization of tonal patterns in music
- Analysis with the Spiral Array
  - MIDI input mapped to Spiral Array: pitch spelling
  - CE tracking: linear filter for memory decay
- Chord and key tracking: nearest neighbor search
- Interactive visualization
  - Real-time rendering (OpenGL)
  - Camera control
  - Autopilot

**Related Work**

- MuSA.RT is the first system combining real-time music analysis and content-based, interactive 3D visualization of tonal structures
- Visualization:
  - Rubato (Garber): off-line music analysis and performance rendering
  - Tonal Landscapes (Sapp): off-line analysis and 2D color maps of tonal patterns
- Real-time system:
  - Continuator (Pachet): music generation system, not multimedia (only sound)

**System Design**

- Main data repository
  - Spiral Array structure
  - Process parameters
- MIDI event processing
  - New custom node
- Rendering
  - (+autopilot)
  - Camera control
  - 4 distinct streams

**Tonal Analysis**

- Use existing MIDI modules (data structures, input/output)
- Spiral Array structure
  - New custom node
- Pitch spelling
  - New custom cell
  - Map MIDI numbers to pitch classes on the pitch spiral
  - Use tonal context
- Real-time CE computation
  - New custom cell
  - Linear filter (low pass)
  - Track two distinct CEs with different memory decay to track chords and keys

**Rendering**

- Use existing OpenGL rendering module
  - White-box module: requires customization
- Visualization:
  - Spiral Array
  - Active triad
  - Chord CE trajectory
  - Active key
  - (Key CE trajectory)

**Camera Control and Autopilot**

- Camera control
  - Use existing game device module
  - Constrain camera position and view angle
- Autopilot
  - Camera position and orientation automatically computed
  - Smooth trajectory (linear filter)
Software Metrics

- Project size estimate: Software Line Of Code (SLOC)
  - MuSA.RT specifics:
    - User interface (MFC)
      - 691 (P) / 487 (L)
    - Custom cells and nodes
    - Exclude general purpose modules
  - Only actual coding concerns algorithms and interface
    - FSF is non trivial to design and code!

<table>
<thead>
<tr>
<th>SLOC</th>
<th>Physical</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>MuSA.RT Op.2</td>
<td>4784</td>
<td>3489</td>
</tr>
<tr>
<td>FSF</td>
<td>2817</td>
<td>2100</td>
</tr>
<tr>
<td>Total 1</td>
<td>7601</td>
<td>5599</td>
</tr>
<tr>
<td>MuSA.RT specific Rendering adj.</td>
<td>3164</td>
<td>2380</td>
</tr>
<tr>
<td>Total 2</td>
<td>3558</td>
<td>2193</td>
</tr>
<tr>
<td>Gain</td>
<td>4975</td>
<td>3856</td>
</tr>
<tr>
<td>(% Total)</td>
<td>64%</td>
<td>64%</td>
</tr>
</tbody>
</table>

System Performance

- Live demo on dual P4 2.8 GHz
- Throughput
  - Rendering: 640x480 @ 25-30Hz
  - Analysis: 30Hz
  - Gamepad sampling 30Hz
- Latency
  - “Looks” immediate
  - Actual measurements: work in progress...
- Possible bottlenecks: MIDI event density, graphics subsystem

Chord and Key Tracking Performance

- Live demo...

System Performance

<table>
<thead>
<tr>
<th>Logical</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSF</td>
<td>2100</td>
</tr>
<tr>
<td>MuSA.RT specific Rendering adj.</td>
<td>397</td>
</tr>
<tr>
<td>Total 1</td>
<td>3586</td>
</tr>
<tr>
<td>Gain</td>
<td>4876</td>
</tr>
<tr>
<td>(% Total)</td>
<td>64%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logical</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rendering adj.</td>
<td>397</td>
</tr>
<tr>
<td>Total 2</td>
<td>1983</td>
</tr>
<tr>
<td>Gain</td>
<td>3586</td>
</tr>
<tr>
<td>(% Total)</td>
<td>64%</td>
</tr>
</tbody>
</table>

Perspectives

- Key and chord tracking methods
- Audio input
- Visualization
- Interaction modalities
- Use as model for larger scale integration experiments