collaborative development of interactive software systems
software architecture for immersipresence

alexandre r.j. françois
visiting assistant professor of computer science
tufts university

research assistant professor of computer science
viterbi school of engineering, university of southern california
Software Architecture

Design, analysis and implementation of software systems
  Improve the flexibility and comprehensibility of software systems
  Address modularization as a design issue
    (higher level than language level)

Explicit system structure
  Technical basis for design
  Provable properties
  Blue-prints for implementation
  Tools for analysis

Project management
  Separation of concerns
  Planning: cost estimation, resource allocation
Immersipresence

• Vision of the Integrated Media System Center
  – NSF ERC in Multimedia, est. 1995-96

• “Combine real world with virtual world”
  – Experience immersion, presence
  – Interact naturally
  – Collaborate through shared virtual/augmented space

• Build systems capable of:
  – Handling video, sound, haptics, etc.
  – Real-time analysis/synthesis (immersion)
  – Low latency (interaction)
Requirements for Immersipresence

- Interoperability
  - Combine research from different fields/teams
- Efficiency
  - On-line, real-time, low latency
- Scalability
  - Performance evaluation and prediction
- General model for distributed asynchronous concurrent processing of data streams
SAI Principles

Time
Volatile vs. persistent data
Asynchronous concurrent processing

Architectural style [ICSE2004]
  High level abstractions
  Hybrid model or more general model?
related work: concurrent computation

process calculi: csp, ccs, acp, pi-calculus
  synchronous message passing
  abstract time
actor model
  asynchronous message passing
  concurrent processing
ptolemy
  heterogenous mixtures of models of computation
  hard real-time systems
  time is an external constraint (add-on)
related work: visual programming

Khoros/Cantata

LabView

Pd

OpenMusic

Max/MSP
related work: adls and uml

architecture description languages (adls)
syntax (graphical) + semantics + tools
domain/style specific: rapide, c2, sadl, wright, etc.
unification efforts: acme, alfa
dynamic systems?
unified modeling language (uml)
object-oriented concepts
collection of loosely related standards
structure? scalability?
primitives & organizing principles

cell
  processing unit (no state)
repository
  shared persistent data
stream
  flow of volatile data
  process dependency
  process trigger
Architectural Middleware

Support architectural abstractions
Pulse, source, cell, etc.

Direct mapping from logical specification to code!

Modular Flow Scheduling Middleware (MFSM)
Open source project: mfsm.SourceForge.net
C++, cross-platform
(GNU compiler)
Base library, functional modules, documentation, tutorials
MFSM Diagram
VisualSAI
CAMSHIFT Tracking

Tracking algorithm: CAMSHIFT
Continuous Adaptive Mean SHIFT (Bradski, 1998)
Mean shift: iteratively find the mode in a probability density distribution (Comaniciu & Meer, 1997)

Perceptual User Interface
Real-time video processing
camshift graph

- Video input
  + input image

- CAMSHIFT processing
  + composite image

- CAMSHIFT parameters

- Image display
architectural refinement: camshift

Video input +input image
CAMSHIFT tracker +new position
Rendering +composite image
Image display

CAMSHIFT tracker node

Logitech QuickCam Pro 4000
IPCam In IPCam Out
Input
IPCam Out
Video Input
Video Render
architectural refinement: camshift

Step I
- Color conversion
  +HSV image

Step II
- Back-projection
  +back-projection image

Step III
- CAMSHIFT
  +new bounding box, size and orientation

Rendering
+composite image

Image display

"Constrains"
loops and time

iterative process => feedback

while((eps>min) && (n<max)){
    kp=f(k);
    eps=g(kp-k);
    k=kp;
    n++;
}

rate

update k

k

+p(k)

f
concurrent processing
barrier synchronization
tracking pattern
feedback
robust
adaptive
visualization
feedback to developer
modular architecture
patterns
facilitate integration
loops and time

iterative process => feedback

```java
while((eps>min)&&(n<max)){
    kp=f(k);
    eps=g(kp-k);
    k=kp;
    n++;
}
```

3d engine event loop => concurrent streams

```java
while(1){
    process_user_inputs();
    process_nw_inputs();
    process_physics();
    process_ai();
    render();
}
```
musa.rt architecture

- MIDI input
  + event
- Process event
- Compute tonal structures
- Spiral array camera and other process parameters
- Poll device
  + command
- Render
- Control camera
esp architecture

MIDI event buffer: current position and velocity parameters for rendering and other processes.

Generate MIDI events.

MIDI output.

MIDI synthesizer.
mimi architecture

MIDI Input
- MIDI input interface
- MIDI input
- Process event
- Current state
- Sample
- Add character

Improvisation Engine
- Oracle and other process parameters
- Improvize
- + character

Performance Engine
- Push
- Play
- Tracks

Visualizaion
- Pulsar
- Render oracle
- Render tracks

Performance
- Push
- Play
- Tracks

MIDI Output
- MIDI output interface
- MIDI output
- MIDI events
- Current state
- Generate MIDI events

MIDI Input interface
- Pulsar

Oracle and other process parameters

Add character

Current state

MIDI output interface

Sample

MIDI input

© arjf 2009
Summary

SAI: Software Architecture for Immersipresence
  Design and analysis of complex software systems
MFSM: Architectural middleware
  Patterns for synchronization

Applications:
  Interactive music systems, Computer vision and graphics systems,
    Distributed Interactive games, etc.

For more information:
  http://iris.usc.edu/~afrancoi
  http://mfsm.sourceforge.net
SAI Properties (1)

Model time explicitly in data and processing
Model modularity
  Separation of concerns
  Scalability
Model concurrent execution (asynchronous)
  Decouple throughput and latency
Model distributed computing
SAI Properties (2)

Facilitate system design
   Intuitive architectural style, based on data streams
   Unified processing model and unified data model
   Design patterns

Facilitate system analysis
   Safety, liveness, etc.

Facilitate distributed development
   Fast integration
   Code reusability

Facilitate system maintenance, modification and evolution
   Change in algorithm and in function
Acknowledgments and Disclaimer

The research presented here was funded in part by:

The Integrated Media Systems Center, an NSF Engineering Research Center, Cooperative agreement No. EEC-9529152.
The Advanced Research and Development Activity of the U.S. Government under contract No. MDA-908-00-C-0036

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the funding agencies.