Using Haskell as DSL for controlling immersive media experiences

FARM 2014
2014-09-06

Outline

- What is LARP?
- What was The Monitor Celestra?
- Technological support systems
- Immersive sound
- Haskell: strengths and drawbacks
- Sound system in action

What is LARP?

- A collaborative storytelling game
- Plays in real time, in a joint physical area
- Players wear costumes, use props
- No spectators: to see is to participate
- All genres
- Geographic variations in style

Nordic LARP

- Nordic LARP style is characterized by eep immersion and player control
- Faithful and complete representation of game world highly valued
- “Railroading” and excessive rules control strongly discouraged

The Monitor Celestra

- Nordic LARP held in 3 repeated games, March 2013.
- Played in the fictional setting of Battlestar Galactica
- The WW2 Destroyer Småland was rented and remodeled to give an immersive impression of a space ship interior

Immersion supported by technical aids

- Laser-cut computer control terminal fronts
- Laser-cut personal dogtags
- Replacing all existing signage
- Visual design
- Designed soundscapes

Soundscapes

- The ship was anchored in Gothenburg harbor: city sounds leaked in
- Full immersion was assisted by creating custom soundscapes on board

Sound System

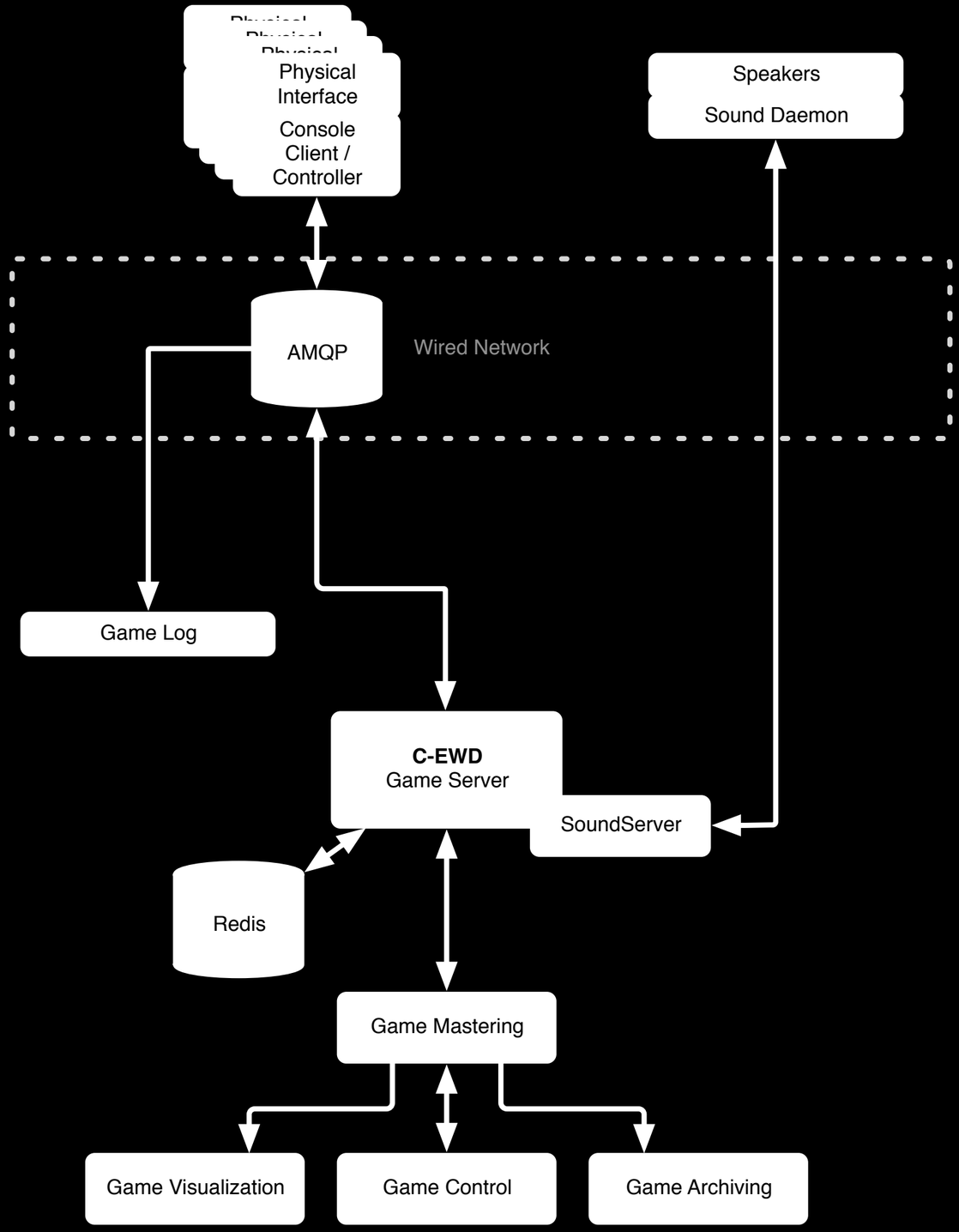
- Custom Build Sound Distribution and synchronization system
- Built to withstand system failure

- Real Time mixing of parameterized ambience creating a dynamic soundtrack for the game
- Creates an ambient feeling of the ship and its state
- enables sound to travel through the ship with millisecond synchronization creating a feeling of localized sound

Sound System – Hardware

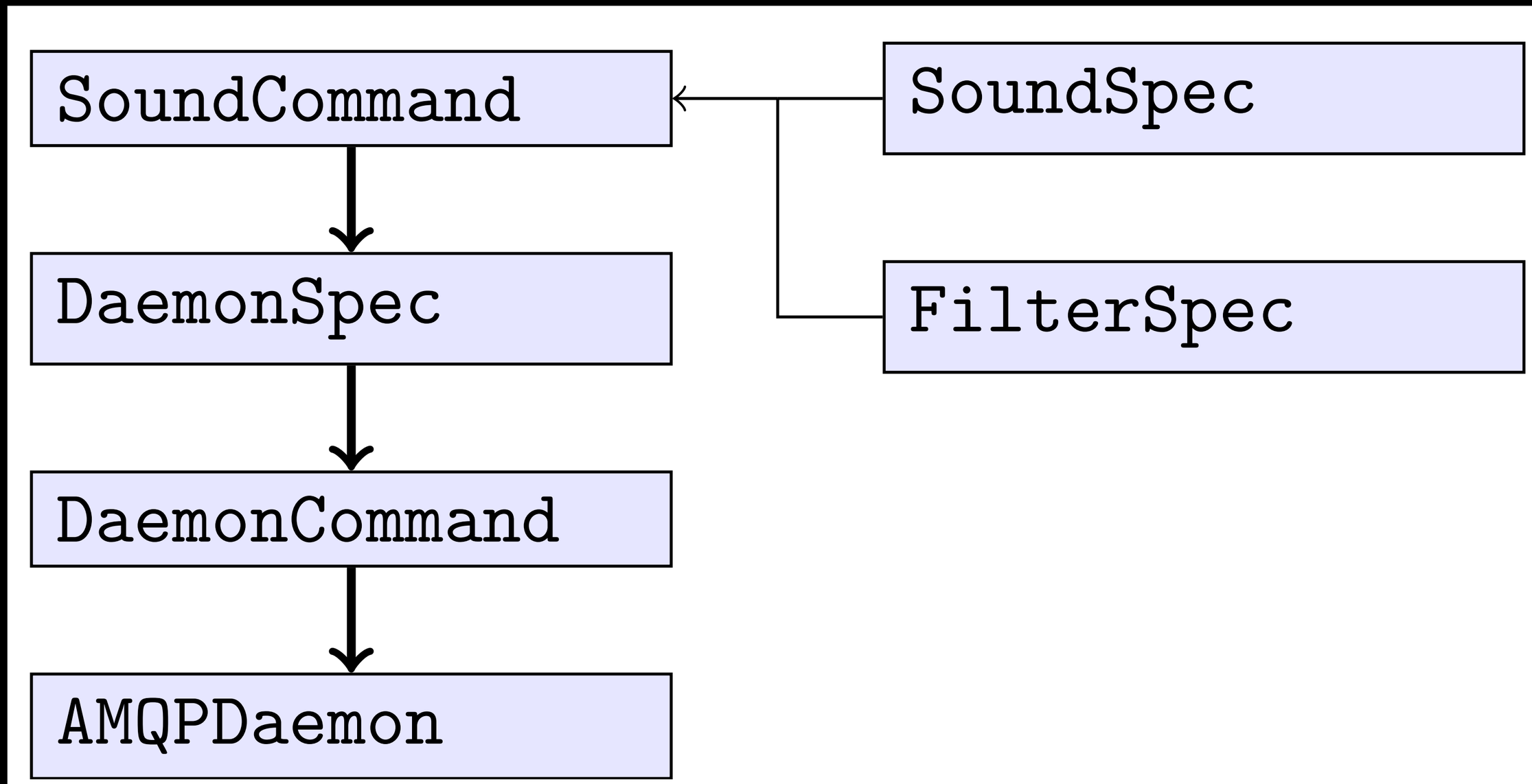
- One dedicated Raspberry Pie for each pair of speakers
- network attached
- real time monitoring

Architecture


Physical Interface
Console Client / Controller
Speakers
Sound Daemon
AMQP
Wired Network
Game Log
C-EWD
Game Server
SoundServer
Redis
Game Mastering
Game Visualization
Game Control
Game Archiving

Types work for us

- Declare datatypes to encode all structures
- Declare translation functions to dig deeper into the communication stack
- Use automatic JSON encoding and parsing


SoundCommand
SoundSpec
DaemonSpec
FilterSpec
DaemonCommand
AMQPDaemon

SoundCommand

- Commands that can be given to the sound specification system
  - Define a sound scape
  - Save / Restore from database
  - Diagnostics
  - Execute specific sound
  - Trigger sound on events
  - Chain commands — Monoid structure

SoundSpec

- Descriptions of Sound Scapes
  - Play, Loop, Stop or Modify a Sound File
  - Crossfade
  - Pick Loudspeaker with indexing
  - Pick Loudness & Left/Right balance
  - Include a delay before command starts

FilterSpec

- Collection of regular expression rules to trigger actions on messages in AMQP queue
- Allows automatic reactions to player devices: “Load Torpedo” automatically creates torpedo loading noises

DaemonSpec

- Translates the Play/Loop/Fade/... commands in a SoundSpec into the primitives used for the lower level sound system:
  Play, Loop, Stop, Change

DaemonCommand

- Wrapper around DaemonSpec that creates JSON messages optimized for parsing by lower level sound system.

AMQPDaemon

- Wrapper to package a DaemonCommand in an AMQP message for delivery to lower level sound system

This is where the demo would have been…

- Discovered yesterday that the surrounding system doesn’t work with the MacOSX stock ruby1.8.
- Not able to show the system in action

Lessons Learned

- Several of our ambitions did not come through:
  - Creative staff never wrote any code
  - Overall system fragile to rebuilds outside exact controlled (version by version) layout
  - Sporadic and untraced performance issues at launch: delays in sound reactions
  - Communication issues between creative and tech groups
    “You need Stereo sound to play it stereo?”
    — discovered after 1 full game round

Lessons Learned

- Other ambitions turned out exactly as hoped for
  - Very quick development and debugging turnarounds
  - Comfortably specified embedded DSL
  - Easy to use Haskell primitives to speed up sound specification

So Say We All