Title
EmStar Software Environment

Permalink
https://escholarship.org/uc/item/0nj8w1pm

Authors
L. Girod
T. Stathopoulos
N. Ramanathan
et al.

Publication Date
2005
EmStar Software Environment

CENS Systems Lab – http://research.cens.ucla.edu

**Problem Description:** Wireless Sensor Network Applications Have Numerous Requirements

**Software Requirements**
- **Algorithms / Sensing**
  - Calibration, Time Synchronization, Task Scheduling, Signal Processing
- **Heterogeneity**
  - NesC / C duplication, Inter-platform Communication
- **Communication / Transport**
  - Not everything is one hop, poor/hard to estimate links
- **System monitoring / Remote Management**
  - Process Control, Logging, Remote Monitoring, State Inspection, Interactive Debugging, Visualization

**Things Break! Always!**
- There will ALWAYS be…
  - Bugs and Unexpected and Transient Behavior
- Trouble is cause by…
  - Multiple asynchronous inter-dependent events
  - Unexpected data and environmental conditions

**Desired Software Properties**
- **Robust**
  - Applications must keep running
  - Can not atomically restart world
  - Partial failure is normal in large distributed systems
  - Softstate is crucial to success of distributed systems
- **Diagnosable**
  - Determine what the problem is
  - In the lab and in the deployments

**Design Choices:** An EmStar Application is Multiple, Event-based Processes

**Multiple Processes**
- Each application is multiple processes
- Provides fault isolation
  - Single component can not take down system
  - Restart individual components
- Provides modularity
  - Clean interfaces
  - Swappable and reusable components
- Encourages softstate design

**Event Driven**
- Each process follows an event based programming model
  - Handles asynchronous inter-dependent events
  - Processes communicate frequently
- Single thread of execution
- Avoid synchronization issues

**The Five Layers of EmStar Provide:**
- **Simulation:** Rapid iteration via real-code simulation tools
- **Module Reuse:** Leverage existing libraries, tools, and services
- **Visibility:** Easily debug/diagnose running systems
- **Robustness:** Keep running despite unexpected failures and bugs

**EmStar Architecture: Five Layers of Service**

**Layer 0:** FUSD
- Low Level IPC
- FUSD: kernel module that allows user-space processes to create device files
- EmStar processes create device files used for IPC
- Devices are the interfaces to a component
- Easy to integrate existing tools

**Layer 1:** Glib
- Handle events on IPC
- Glib provides EmStar with an event-based programming interface
- No EmStar processes poll or block
- Single thread of execution
- Low level events: reads/writes to device file
- High level events: timer, sending receiving packets, communicating state

**Layer 2:** Device Patterns & Libraries
- IPC mechanism for a variety of interactions
- Common usage patterns for IPC in apps
- Libraries create interfaces that follow the common patterns
- Reads and writes from devices abstracted according to the pattern
- Usage patterns follow client-server model

**Layer 3:** Existing Modules, Services
- Existing useful components for applications
- Reuse existing modules and services
- Many already implemented components
  - neighbor discovery
  - link-quality estimators
  - time synchronization
  - high-level state synchronization

**Layer 4:** Extra Tools
- Help run, maintain, and debug application
- Emrun/Ensim: manage individual nodes/simulations
- Emview: visualization tool
- EmTOS: NesC integration libraries
- EmStar Web Server: customizable access to devices files through the web
- FUSdnet: exports devices files over network interfaces

**Future Work**
- Increased IPC & FUSD Performance
- More accurate radio & sensor simulation
- Expanding routing & transport Services
- Integration with numeric & signal-processing tool-kits

---