Title:
Em View: The Em* Visualizer

Author:
Lewis Girod; Alberto Cerpa; Henri Dubois-Ferriere

Publication Date:
01-01-2003

Series:
Posters

Publication Info:
Posters, Center for Embedded Network Sensing, UC Los Angeles

Permalink:
http://escholarship.org/uc/item/7q60k5d3

Additional Info:
BibTex @misc{ title={Em View: The Em* Visualizer}, author={Lewis Girod, Alberto Cerpa, Henri Dubois-Ferriere}, abstract={Em View is an extensible visualization system that integrates to emstar-based systems, simulators, and emulators. It can also be connected via gateways and other means to other systems. Em View grew out of a need to develop a unified visualization system to replace a host of individually developed single-application visualizers. Thus, Em View is designed specifically to visualize a wide range of applications from a single reusable framework. A highly customizable interface enables multiple visualization schemes to coexist in a single display, while a simple API significantly reduces the coding required to develop new application specific visualization modules.}, url={http://research.cens.ucla.edu/pls/portal/url/item/F2ECE1DABC88CD79E0306180528D62BB}, year={2003}, }

Abstract:
Em View is an extensible visualization system that integrates to emstar-based systems, simulators, and emulators. It can also be connected via gateways and other means to other systems. Em View grew out of a need to develop a unified visualization system to replace a host of individually developed single-application visualizers. Thus, Em View is designed specifically to visualize a wide range of applications from a single reusable framework. A highly customizable interface enables multiple visualization schemes to coexist in a single display, while a simple API significantly reduces the coding required to develop new application specific visualization modules.
Introduction: Visualization for Development of Distributed Embedded Systems

Visualization Critical for Distributed Systems
- Concurrent Processes
  In a distributed system, algorithms run concurrently on many independent nodes, and interact through communication channels
- Real Time Dynamics
  Debugging the interaction of distributed participants in an algorithm requires attention to real-time dynamics that reveal conditions that result in instability
- Real Time Condition and Exception Reporting
  Capturing the behavior of a large distributed system requires visualization of infrequent exceptions and condition state from a large collection of nodes

Embedded vs. Distributed Visualization
- Embedded Visualization
  - Embedded visualization uses the same communication channels used by the application to report debugging information from a deployed system
  - Requires more development time to optimize the messaging, and often can’t provide reliable information on real-time dynamics
- Distributed Visualization
  - Distributed visualization uses a separate debugging backchannel to gather data from a system in the lab, or an instrumented deployment.
  - Message reporting optimized for low development effort

Requirements for EmStar Visualization
- Extensible, Modular Support for Many Applications
  - Minimize development cost of adding visualization support for new algorithms, applications, and system components
  - Enable easy integration of visualization displays for different algorithms, applications, and components; avoid mutual exclusion
- Support for Many EmStar Modalities
  - EmStar can run the same code on a simulator, clustered simulator, ceiling emulator, portable array, or on actual distributed nodes.
  - One visualization system must transparently handle all cases

What EmView Doesn’t Do
- Not an Embedded Visualization Solution
  - EmView solves the “distributed” case, assuming a high-bandwidth debug backchannel.
  - Can leverage gateways that relay embedded debug info to backchannel.
- Doesn’t (Yet) Support Packet-level Protocol Visualization
  - EmView is trying to display real-time state gathered from all nodes, for example live neighbor state, routing state, timesync relations, etc.
  - Packet-level protocol visualization would be useful, but hard to see how to visualize it between more than a few nodes (e.g. event diagram)

Proposed Solution: EmProxy Status Protocol + EmView Modular Visualization Engine

How To Use EmView
- Instrument your Component or Algorithm
  First, identify the state variables and conditions in the component that you want to visualize. Then expose those variables using “status devices”, or use status_reflector. This state will then be visible to EmProxy.
- Add a Module to EmView to Request Component Data
  Create a new module that will visualize the state of your component. This module will define an EmProxy request specifying your new status devices, and a handler to process that data.
- Parse Component Data and Submit it to the Engine
  The handler function will then parse the data as it arrives. (Usually it is easiest to use binary structs in order to minimize the complexity of parsing). After parsing, any relevant data is submitted to the EmView core to be rendered. Note that the module code does not actually render data, rather just submits it to the core in an abstract form, such as a string or numeric value, or a link from one node to another.
- Run and Configure EmView
  In EmView, the request and rendering of visualization data is controlled at runtime. Currently, modules can be activated and deactivated individually; each module has a “default” rendering policy that is activated by turning on the module, overriding any conflicting policies. Although the EmView core supports more or less arbitrary linkage of data elements to components of the visualization, currently there is no GUI interface that allows complete flexibility.

Using EmView

EmView Node Components

DataFlow in EmView

App-specific EmView Modules

Data passed to the EmView core can be assigned at runtime to be rendered in specific “slots” in a node icon. Some of these slots are shown in the diagram below ▼

Three Examples of EmView Modules:

- Visualization of neighbor discovery and linkstats state. Green links are good, asymmetric links. Dotted red links are bad, asymmetric links.
- Visualization of Vortex Monitoring Sink. Sink is colored solid, and the sink trees are color coded according to sink.
- Visualization of multi-hop propagation of “global time” from the node with the red flag.

Note: A and B are visualizing an emulation running on the ceiling array, while C is running entirely in simulation.