EmView: The Em* Visualizer

Lewis Girod, Alberto Cerpa, Henri Dubois-Ferriere
CENS Systems Lab – http://lecs.cs.ucla.edu

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

Problem Description: An Extensible Visualization System for EmStar

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

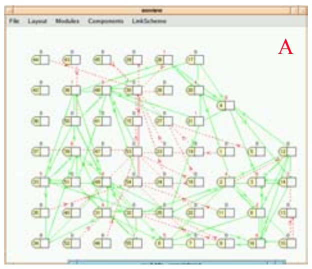
- **Not an Embedded Visualization Solution**
  - EmView solves the "distributed" case, assuming a high-bandwidth debug backchannel.
  - Can leverage gateways that relay embedded debug info to backchannel.
- **Doesn't (Yet) Support Packet-level Protocol Visualization**
  - EmView is trying to display 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)

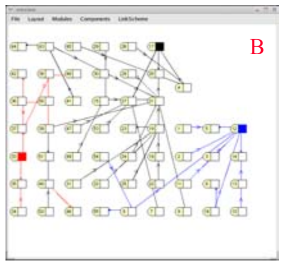
Proposed Solution: EmProxy Status Protocol + EmView Modular Visualization Engine

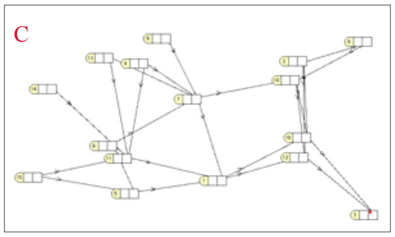
How To Use EmView

- **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.

Three Examples of EmView Modules:



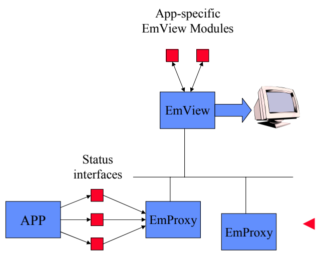




- Visualization of neighbor discovery and linkstats state. Green links are good, symmetric links. Dotted red links are bad, asymmetric links.
- Visualization of Voronoi Diffusion Sinks. Sinks are colored solid, and the sink trees are color coded according to sink.
- Visualization of multi-hop propagation of "global time" from the node with the red flag.

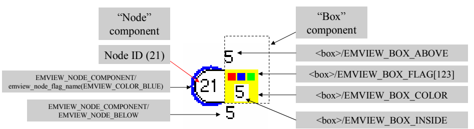
Note: A and B are visualizing an emulation running on the ceiling array, while C is running entirely in simulation.

Using EmView



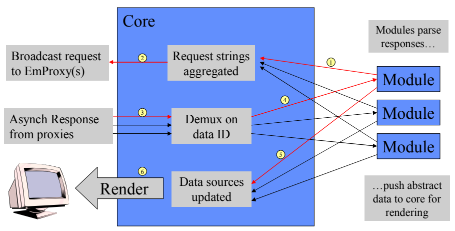
EmView Node Components

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 ▼


"Node" component
Node ID (21)
EMVIEW_NODE_COMPONENT/ emview_node_flag_name(EMVIEW_COLOR_BLUE)
EMVIEW_NODE_COMPONENT/ EMVIEW_NODE_BELOW
"Box" component
<box>/EMVIEW_BOX_ABOVE
<box>/EMVIEW_BOX_FLAG[123]
<box>/EMVIEW_BOX_COLOR
<box>/EMVIEW_BOX_INSIDE
21
5
5
5

◄ Red boxes represent integration of visualization into an application
Shows the sequence of interaction of a module with the EmView core ►

DataFlow in EmView


Core
Broadcast request to EmProxy(s)
Request strings aggregated
Asynch Response from proxies
Demux on data ID
Data sources updated
Render
Modules parse responses…
Module
Module
Module
…push abstract data to core for rendering
1
2
3
4
5
6