Institute for Software Integrated Systems
Vanderbilt University

Experience in Developing Model-Integrated Tools and Technologies for Large-Scale Fault Tolerant Real-Time Embedded Systems

Presented by
Steven G. Nordstrom

Overview

Tools, Techniques, and Software for building large-scale, real-time, physics data systems

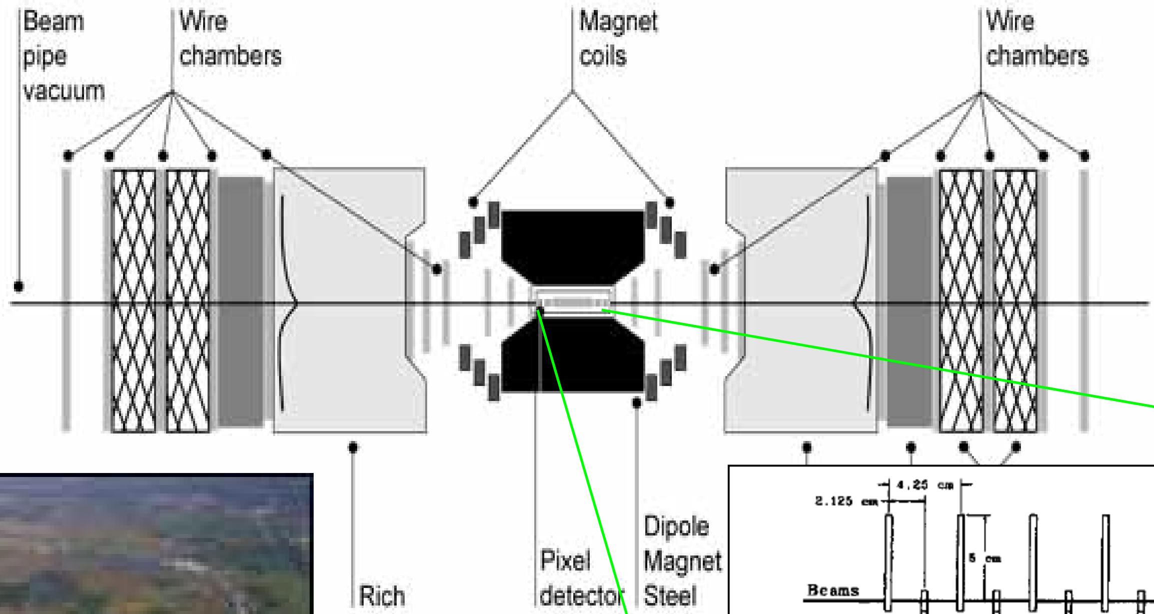
- HEP, BTeV, and RTES
  - Tremendous computational needs
  - High cost of plant operations
- Forces that drive HEP computing needs
  - Results of experiments drive science
  - Systems should be customizable by non-programmers
- Prototype demonstration system
  - Small Scale: ~16 embedded processors
  - Model based techniques for design, maintenance, and runtime failure management
- Lessons learned

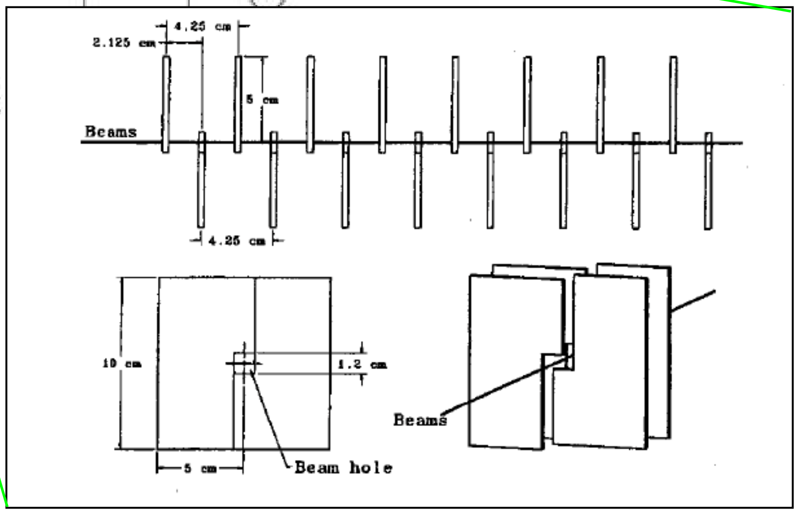
BTeV RTES Team NSF/ITR Supported

- RTES (Real-Time Embedded Systems) collaboration formed to address design, integration, and fault tolerant issues for BTeV
- FermiLab
  - Building BTeV Trigger Hardware
  - Domain Experts, Define Goals, Constraints, etc.
- UIUC
  - ARMOR, Application level fault tolerance, Fault Tolerant Middleware
- Syracuse & Pitt
  - Very Lightweight Agents, Diagnostics, Load Balancing
- Vanderbilt
  - RTES Lead (Physics)
  - Design Environment, System Synthesis, System Integration, Prototype Hardware

High Energy Physics

FermiLab Tevatron





BTeV

BTeV Computations

- Interactions at Level 1 are occurring every 132ns at around 200 KB/event, which is ~1.5 TB/s
- BTeV Experiment requirements in the realm of RTE systems
  - Maximum resilience to failure due to expense of starting and stopping experiments in a particle accelerator
  - Large scale real-time embedded system of ~2500 Level 1 DSP nodes, ~2500 Level 2 and Level 3 Processing nodes (off-the-shelf Linux machines) with minimal redundancy in hardware (project management states redundancy <= 10% of resources)
  - Integrated tools to manage the design, development, deployment and support of said RTE system

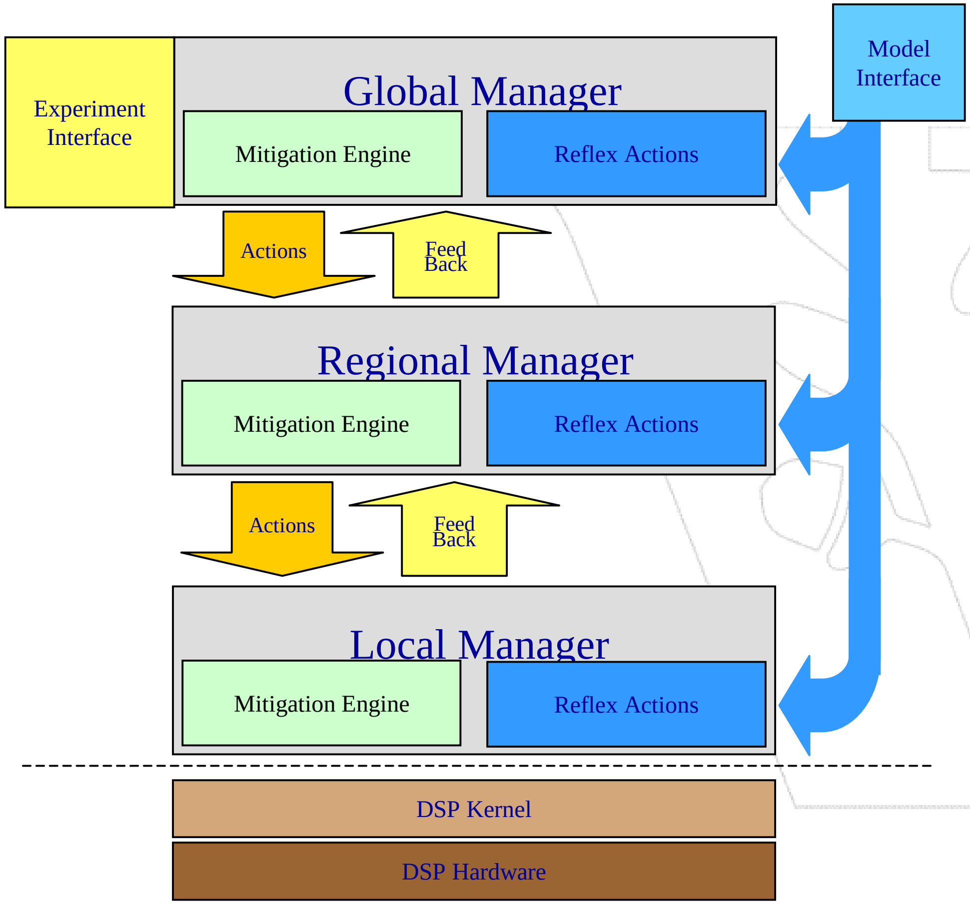
BTeV RTE System

- Should allow specification of hardware, application data flow, and system wide failure management behavior
- Support automated generation, configuration, and deployment of the runtime environment
- Should allow fault diagnosis and adaptation during runtime and should support post-mortem analysis
- The tools should provide the ability to rapidly model, synthesize, and deploy a variety of systems and strategies for high energy physics research purposes

Demonstration System

- Implement a demonstration trigger, that
  - Generates and distributes simulated physics interactions
  - Applies computation to every interaction
  - Demonstrates sustained computational performance
  - Maintains functional integrity for long periods of time
  - Is dynamically reconfigurable, maintainable, and evolvable
- Create fault handling infrastructure capable of
  - Accurately identifying problems (where, what, and why)
  - Compensating for failures (shifting loads, *changing thresholds*)
  - Automated recovery procedures (restart / reconfiguration)
  - Accurate accounting for display and post-mortem analysis
  - Being easily extended (capturing new detection/recovery procedures)
  - Policy driven monitoring and control

Runtime Environment

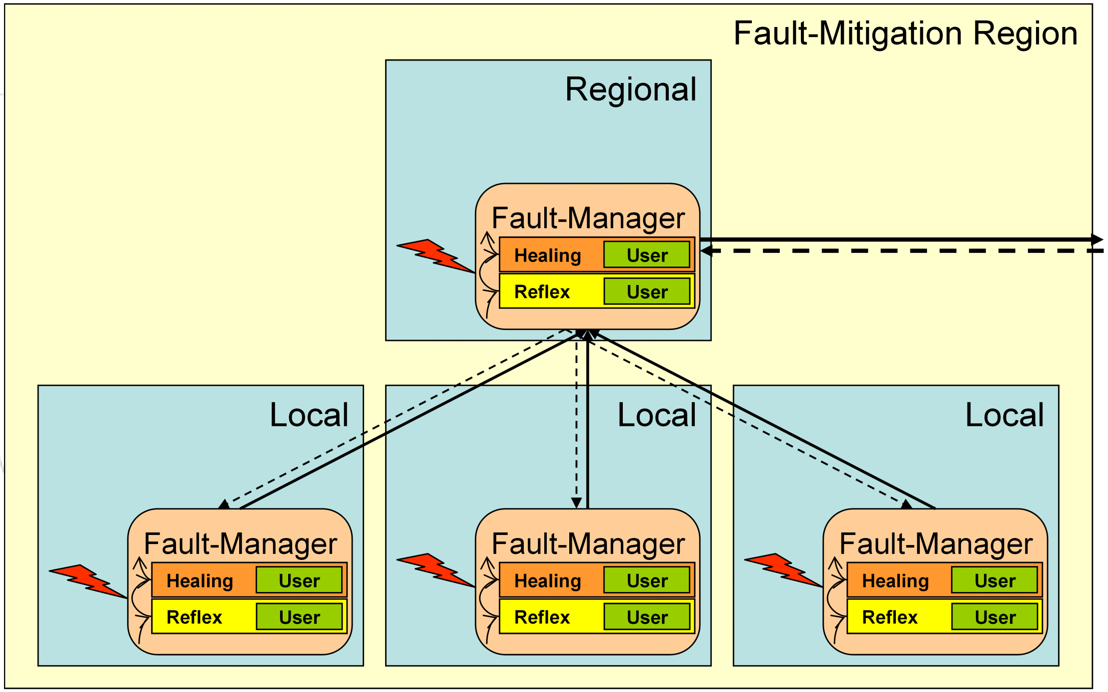


To demonstrate mitigation behaviors in a running system, the runtime environment embodies the following features:

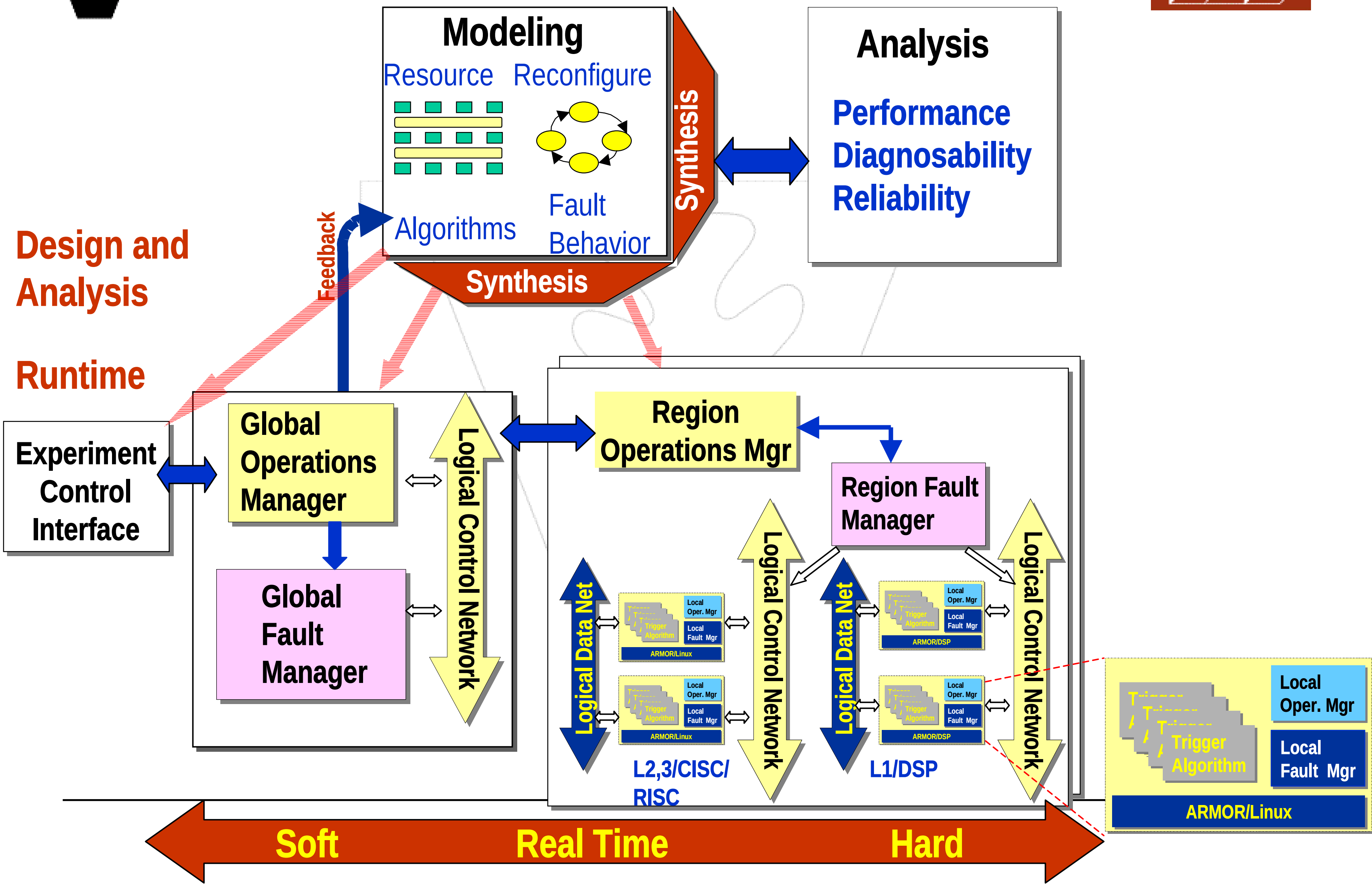
- Fault Mitigation API: This provides a ‘standardized’ interface to the fault adaptation functions of the runtime
- Fault Mitigation Messaging infrastructure, to pass fault detection status messages and fault mitigation actions across the network
- Fault Mitigation Kernel: providing the fault message routing system, processor scheduling, process and communication fault adaptation, etc.
- Fault Monitoring capability
- Fault-Injection: the fault injection system is used to simulate hardware or software faults to test the fault behavior

Fault Handling

- ‘Reflex Action’:
  - Simple, (modify current state/structure only)
  - Rapid, (milliseconds-seconds)
  - Real-Time, Guaranteed Response Time,
  - Sub-Optimal
  - Handle a Single Failure (any component)
- ‘Healing’
  - Re-Evaluate Resources & Tasks
  - Re-Balance/Re-Allocate Resources
  - Recover Failed Resources (After Testing)
  - Generate New Reflex Actions

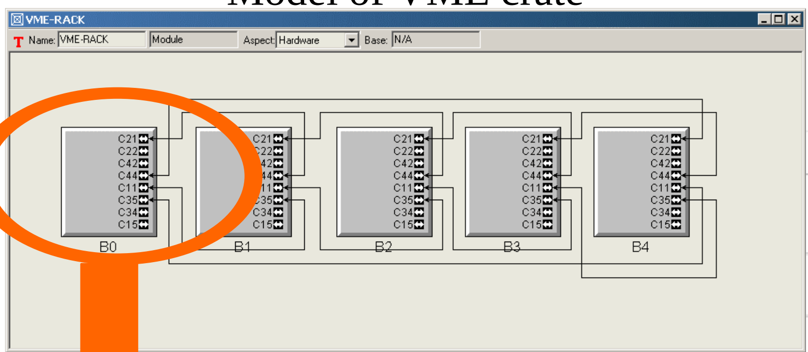


Architecture

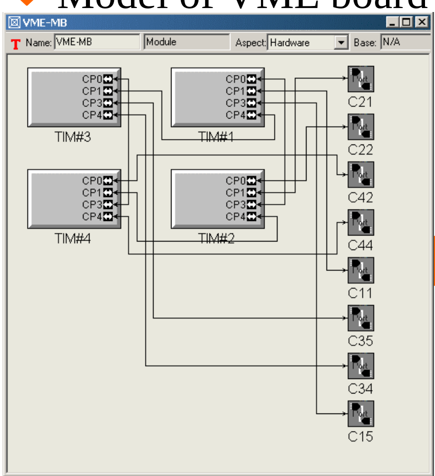


Resource Models

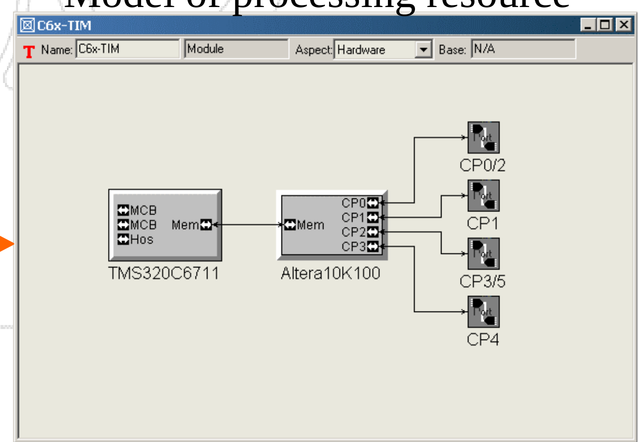
Model of VME crate



Model of VME board



Model of processing resource



- Graphical representation of physical hardware layout
- Hierarchy of models used to abstract complexity
- System resource allocation determined by hardware references within fault mitigation models

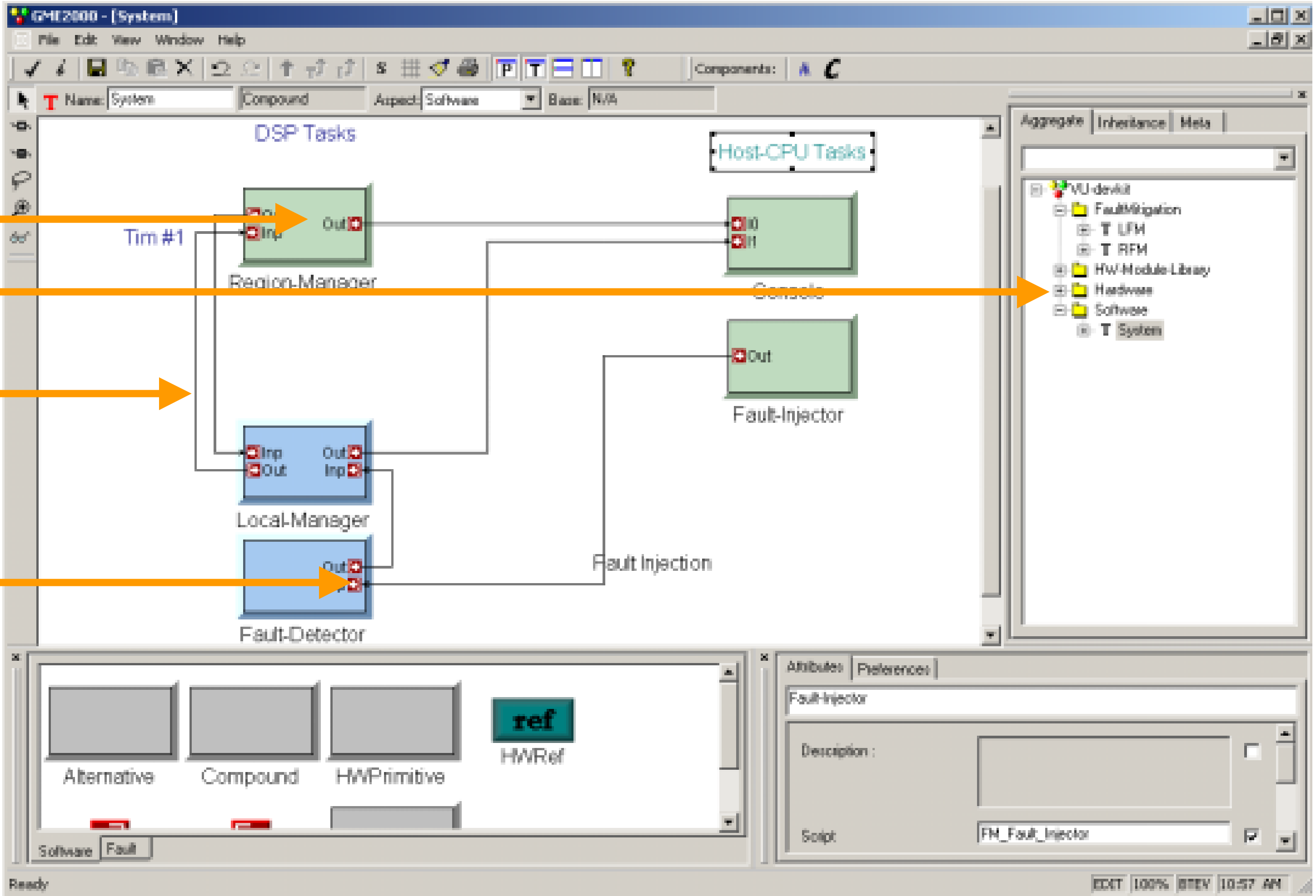
Algorithm Models

Processes

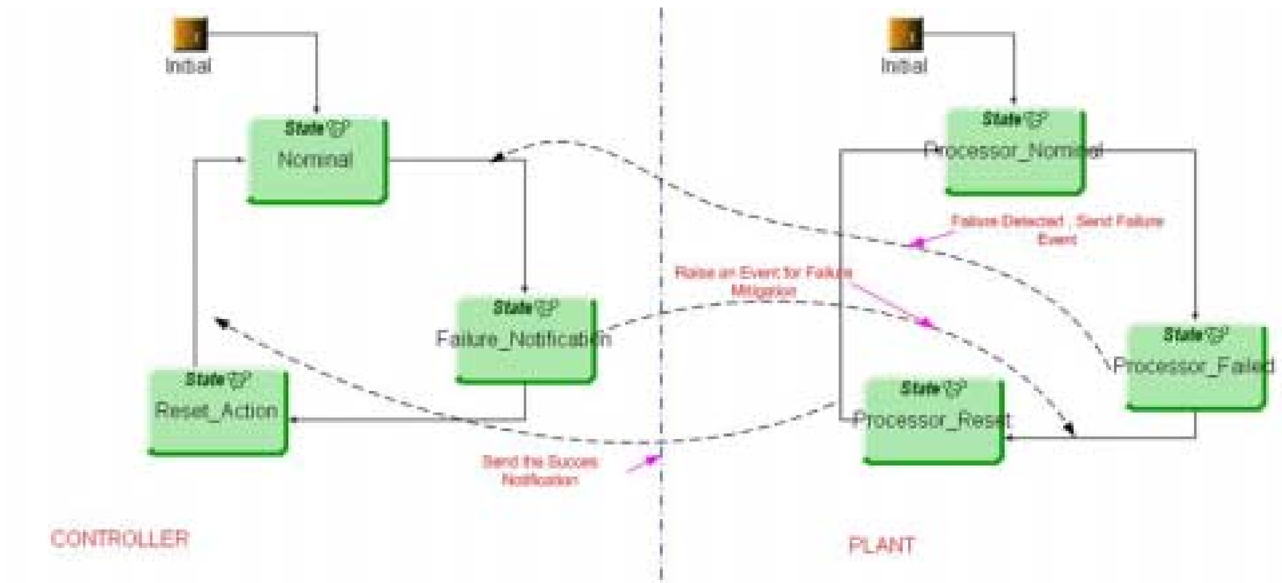
Hierarchy

Information Flow

Interfaces

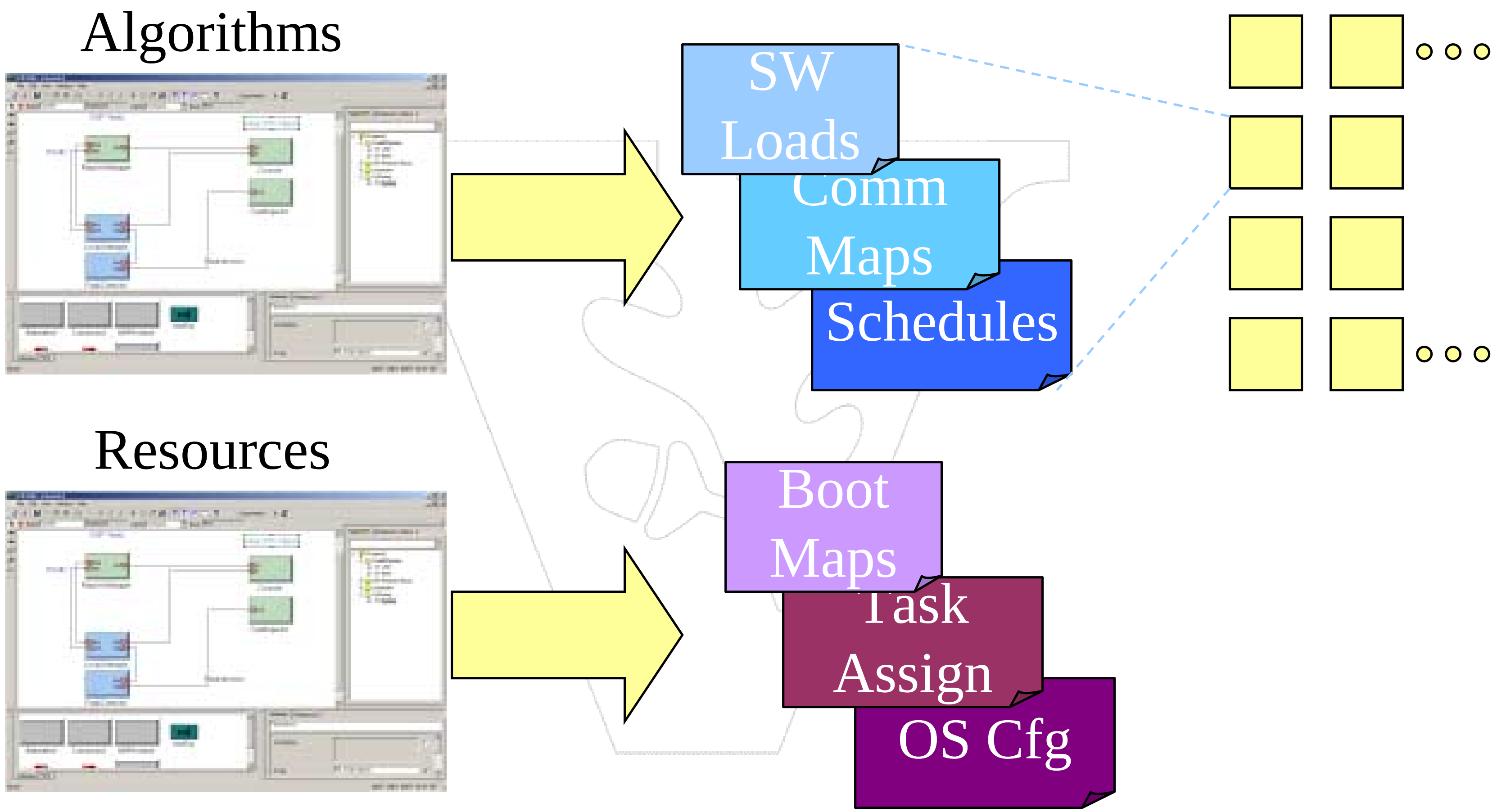


Fault Behavior Models

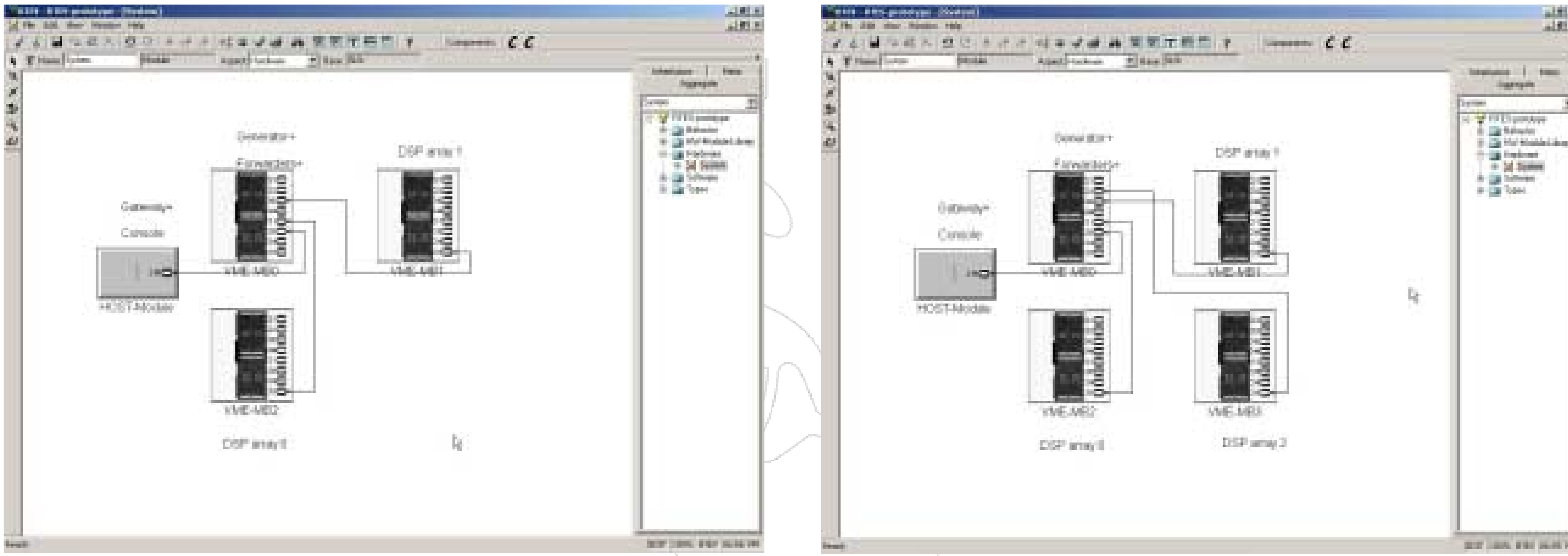


- Models describe behavior of fault mitigation entities using statecharts-like notation
- Actions attached to transitions may call Fault-API methods

System Generation

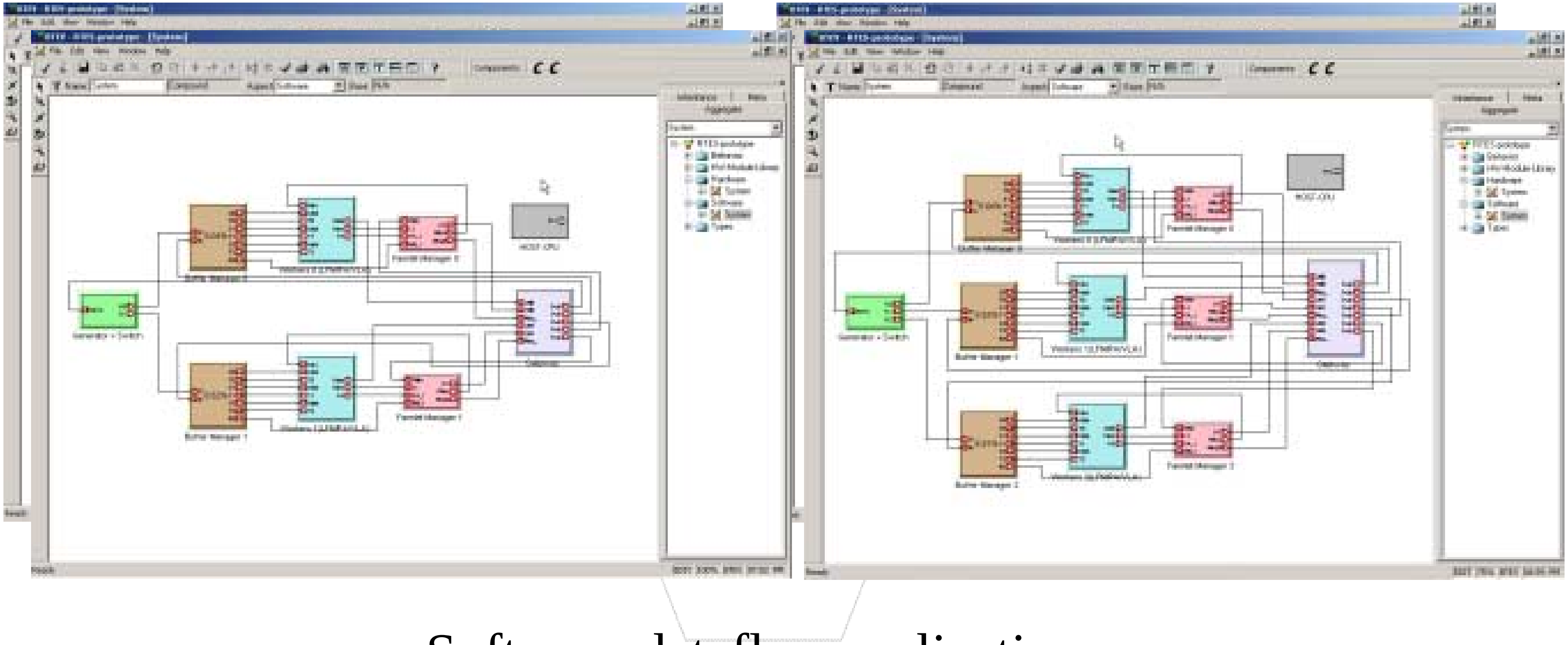


Scaling Models



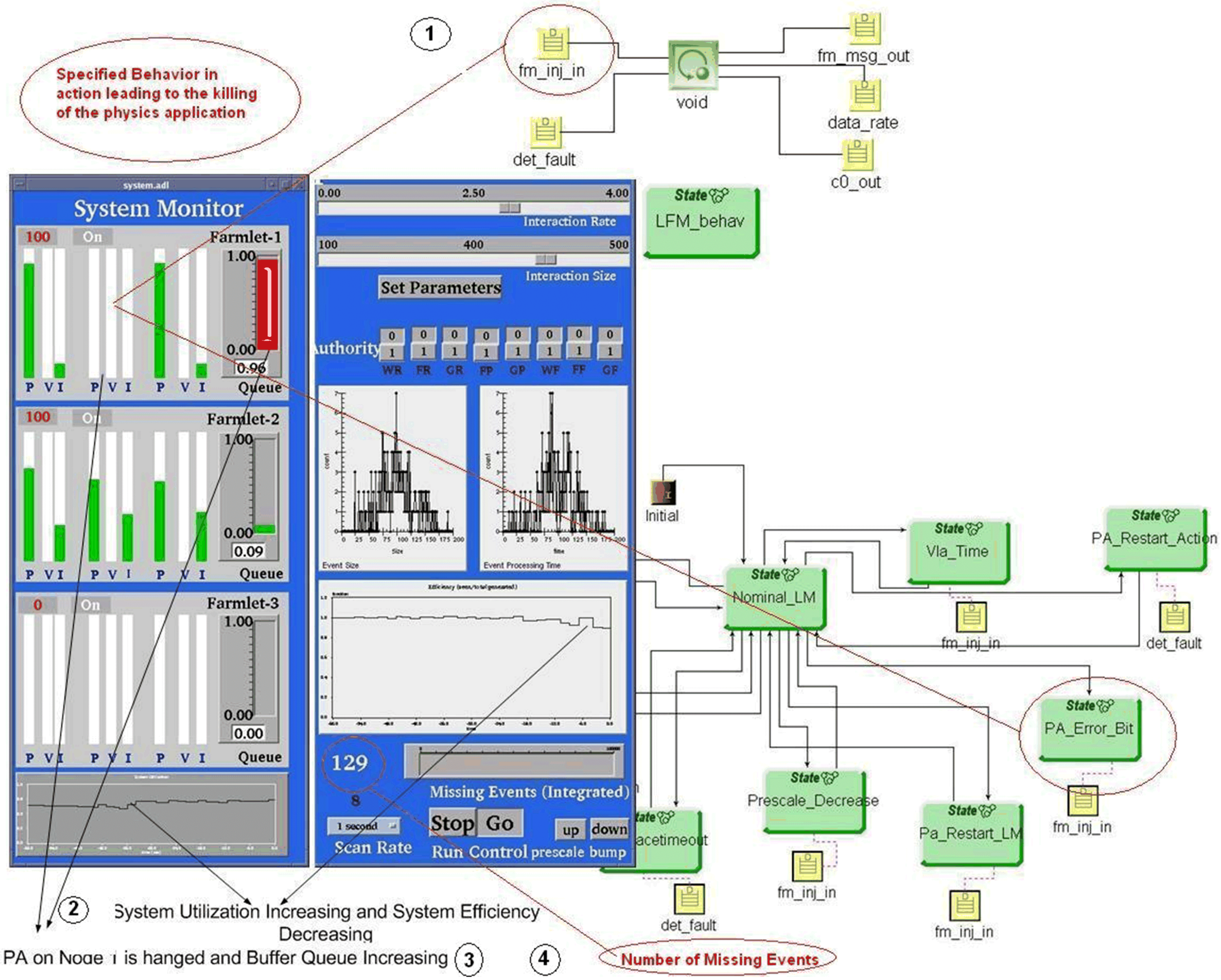
Modification of hardware resources model

Scaling Models

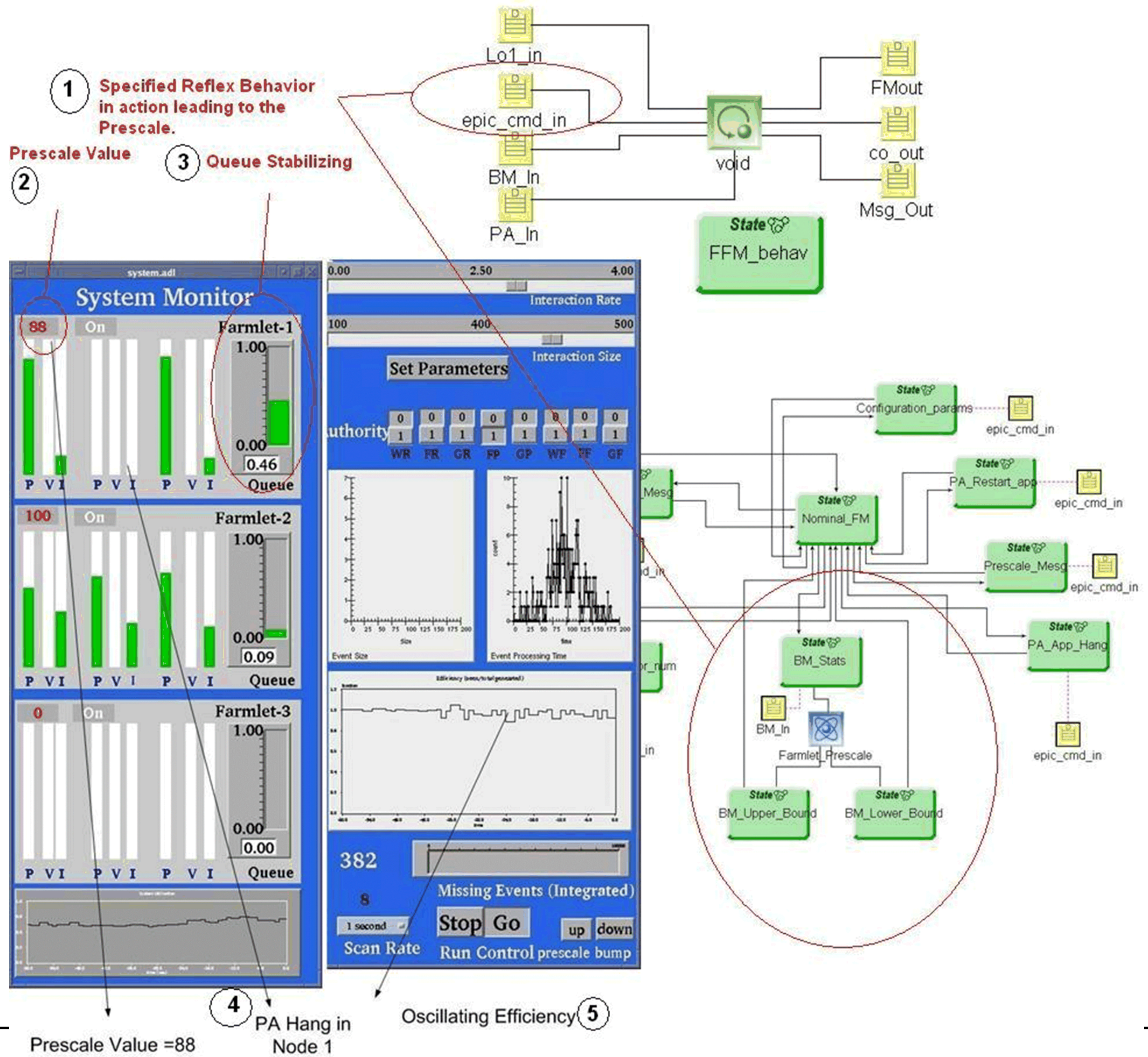


Software dataflow replication

Failure Scenario 1



Failure Scenario 2



Conslusions

- Demonstration system presented at SC2003
  - Design tools used to easily scale system from 4-8-16 nodes
  - Multiple failure strategies modeled and tested and analyzed
  - Application, hardware, and test data failures were demonstrated
- Lessons Learned
  - Our domain specific language for modeling fault manager behavior is not sufficient for a fully analyzable model
  - Scalability challenges still exist in visualization and navigation of extremely large models using this environment
  - Versioning and partitioning of large models is crucial for multiple designers working with system models