Overview Tutorial

www.csim.com

July 6, 2023

chein@csim.com

Outline

- Session 1: (Day 1, 9:00-12:00)
  - Simulation Modeling Concepts
  - CSIM Overview
- Session 2: (Day 1, 1:00-4:30)
  - Installation and Set-up
  - Tool Usage and Techniques
  - Hands-On Examples
- Session 3: (Day 2, 8:30-11:30)
  - Core Performance Modeling Library
  - General Blocks Library
  - Multi-Simulation Interface (MSI), HLA
  - Development Techniques / Workshop
- Session 4: (Day 2, 12:30-4:30)
  - LAN / WAN IP-Network Modeling
  - Visualizations
- Session 5: (Day 3, 8:30-11:30)
  - Processor System Modeling
  - Modeling Workshops
- Session 6: (Day 3, 12:30-3:30)
  - Scenario Modeling

CSIM Overview

– Simulation concepts - virtual prototyping

– System modeling examples

– CSIM tools and their flow

– Structure description - graphical editor (GUI)

– Behavior description - language constructs

– Model Attributes

– Simulator - building + running simulations

– Analysis tools - viewing results

– Application libraries and domains

Why Simulate ?

System Challenges:

- Modern systems are too complex to manually predict or understand all potential issues
- Difficult to comprehend all system and mission interactions/sequences
- Integrations occur late in project cycles –> problems found too late
- Detailed implementations take too long, cost too much to get wrong
- Need comprehensive prototyping, early and often
- Systems are too large to run detailed models within project schedules
- Ad hoc models cost too much, lack credibility
- ***Need efficient rapid virtual prototypes***

(Yet, many challenges to system modeling)

Reasons to Simulate:

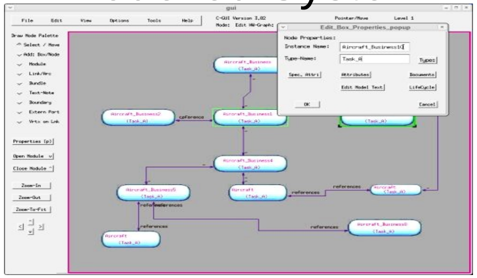
- Network or Architecture - Evaluation, Optimization, or Selection
- Functional allocation & optimization
- Hardware / Software mapping & scheduling
- Risk reduction: logic, timing & performance issues
- Customer prototyping & demonstrations
- Early integration & testing – Find & address risks early

Simulator Requirements

System Modeling & Simulation Requires Tools:

- Arbitrary modeling Abstraction & Detail of:
  - Functions, behaviors, values
  - Structure / Architectures / Interfaces
  - Timing relationships, sequences, delays
- Easy to use, rapid model construction, Graphical Diagram Editor
- Available model libraries, Enable rapid *Model Re-Use*
- Support multiple domains (Networks, Logic, Electrical, Work-flows, ...)
- Platform independent / portable
- Modeling language same as embedded system (C)
- Standards based – C, XML, HLA, ...
- Rich visualization capabilities
- Must be validated, proven, trustworthy, efficient, scalable, fast

Modelled System



Real System

Overview

What is CSIM?

– A <u>general-purpose</u> system simulation environment
  • Discrete event simulator – simulates time & sequences
  • Hierarchical block diagrams connect models
  • Model functions are described in standard C language.
  • Model integrator – manages concurrency, time coordination

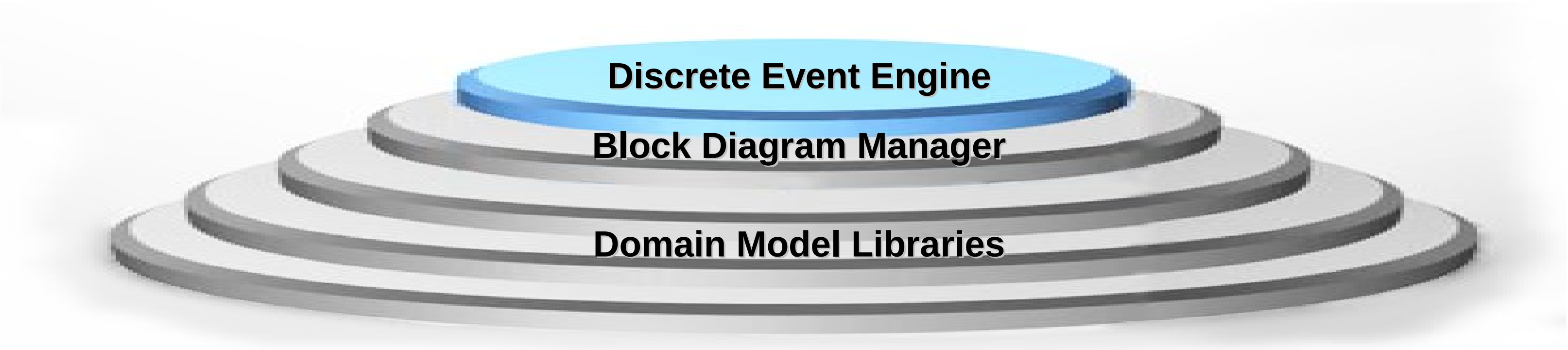
- Quickly model & test complex systems: Networks, Tasks, Architectures, Logic, ...

- Supports system trades: Performance, rates, latency, bandwidth, ...

– Both Graphical and Textual development methods.

– Several domain Model-Libraries speed system modeling

- Supports multiple abstraction levels

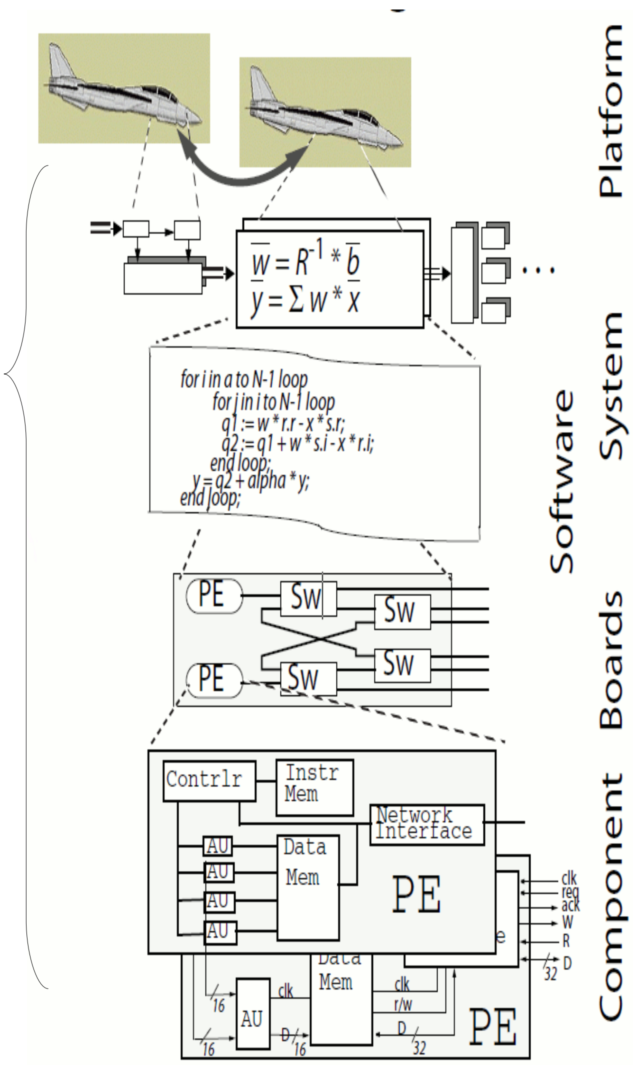


Modeling Concepts

- *Abstraction* versus resolved *Detail*
  - Speed, scalability
  - Visibility
  - Accuracy vs. Precision

- High Abstraction is lower Detail, & vice verse
- A given system, or component, may be described at many different abstraction levels
- A given model may be composed out of a combination of lower level models
- Detail does not imply accuracy

- Before modeling a system, consider:
  - What are your goals ?
  - System concerns ?
  - What do you need to investigate ?
  - What should models contain ?
  - How to test your system & models ?



Tools must support multiple abstraction levels.

The Power of Abstraction

CSIM

Comparative Simulation Speeds

Simulation	Simulated Time	Host CPU Time	Equiv. Processing Rate (/Sec)
Abstract Arch. Model 72-node Network	60-Mins	0.4-Min	12,856,500,000
Performance Model 24-Computer Network	5.0-Secs	3-Mins	28,570,000
Abstract Behavior 6-Computer Network	5.0-Secs	7-Mins	3,081,000
Detailed Model of 1 Computer	5.0-mS	12-Hours	5

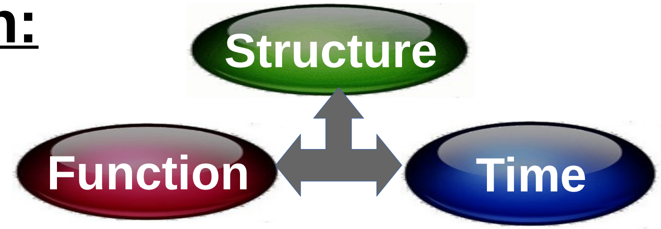
(*All were CSIM simulations, except last, which was VHDL.)

Simulations predicted performance to within 7% accuracy of final system.

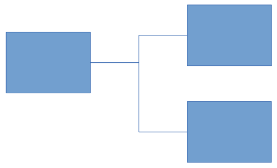
Modeling Concepts

CSIM

Three Axes of Description:



- **Structure** or Architecture
  - Instantiates the components
  - Shows how the components are inter-connected
  - Displayed in Block Diagrams
  - Can have hierarchy (more detail later)



- **Functionality**
  - What a component does, or how it does it
  - Described in equations or code

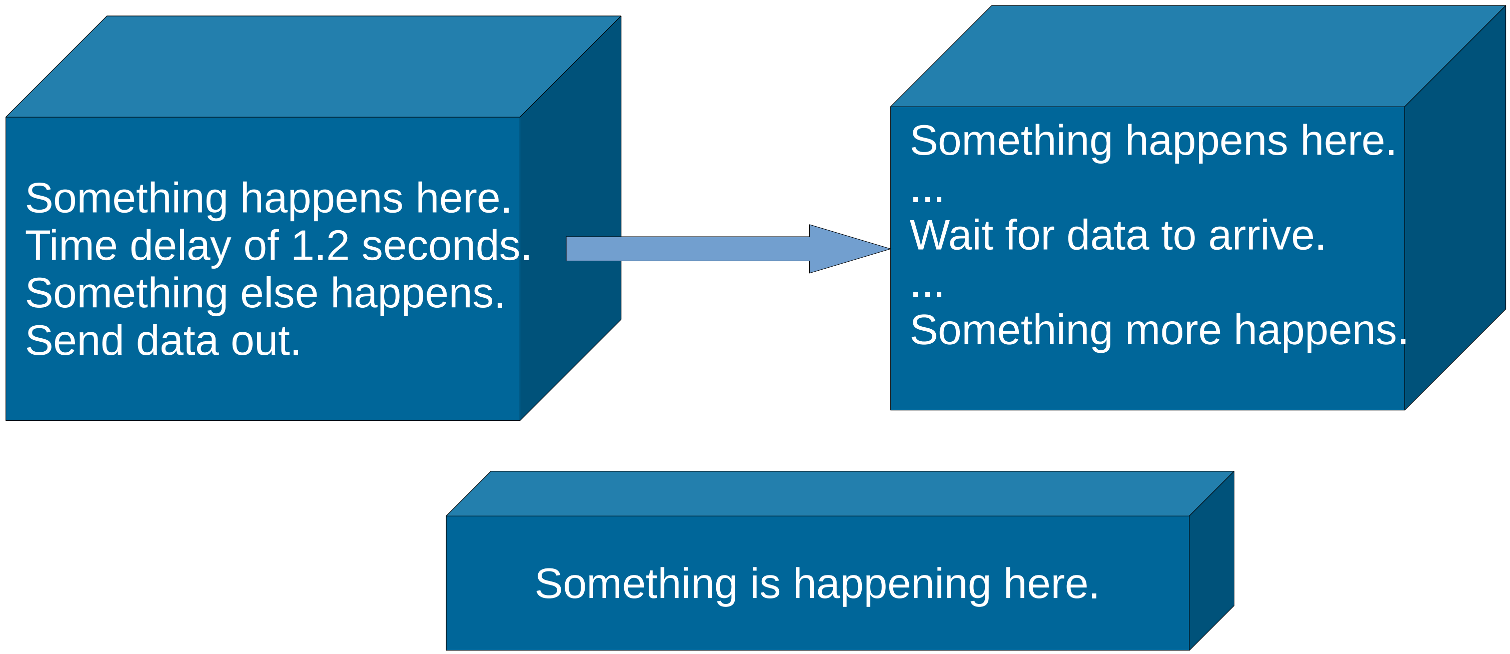
`A = B + C;`

- **Time**
  - Delays, rates, relationships, sequences

`DELAY( 20 mSec );`

Behavior = Time + Function

Simple Example of a System Model



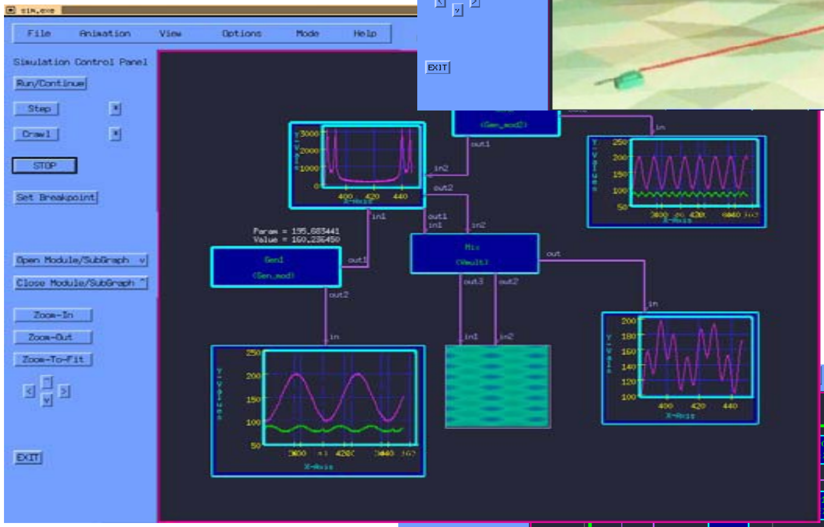
Can you identify:

- Structure ?
- Timing information ?
- Functionality ?

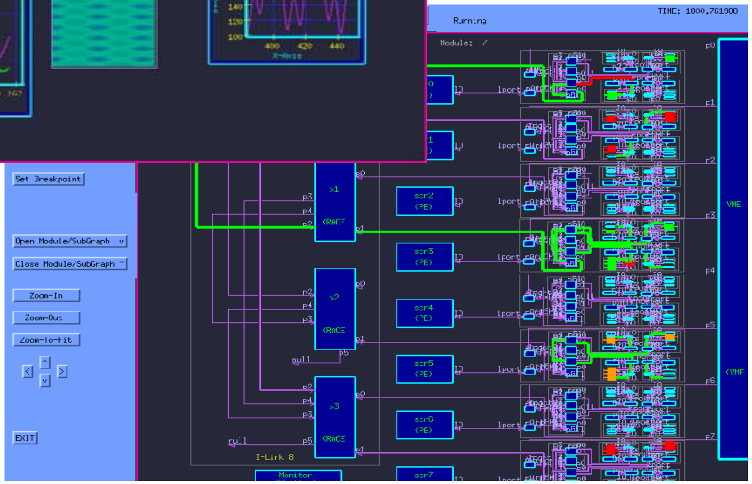
Example of multiple abstraction level modeling

- Multi-Vehicle scenario resolves interactions between platforms within a realistic mission scenario

- Algorithmic level model resolves functions within the vehicle systems above, but not computing system details



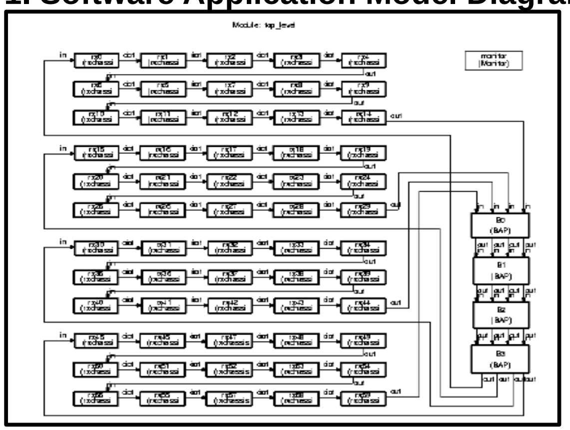
- Multi-processor computer architecture simulation resolves the hardware, software, and data transactions through time to accomplish the above algorithms



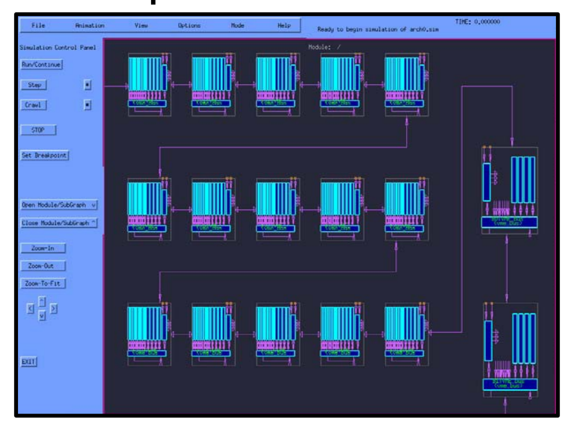
Example Computer Hardware & Software Model

- Networks of processors can accelerate difficult computing tasks
- But coordinating software & data on complex networks is difficult
- CSIM provides tools to understand & optimize parallel systems
  - (1) Model *Software Applications* separate from (2) *Physical Architectures*
  - (3) Simulate the Software models executing on the Architecture models
    - (Hardware / Software Co-simulation)
  - (4) Visualize the performance, identify issues, improve the system

1. Software Application Model Diagram

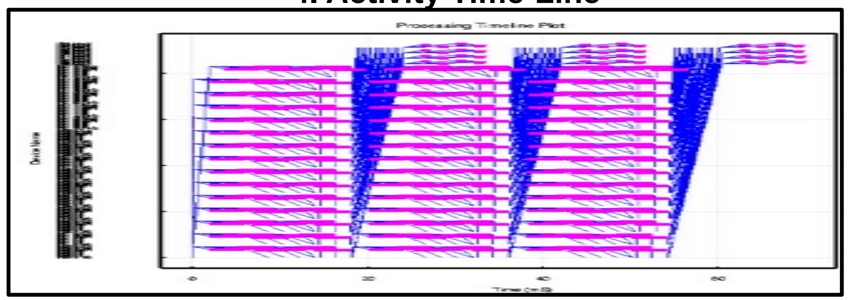


2. Multi-Computer Hardware Architecture Model



3. Simulate

4. Activity Time-Line



Terminology

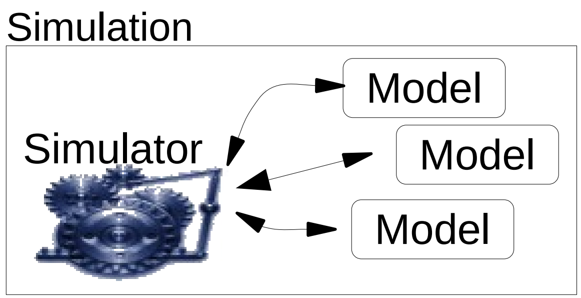
Model – (noun) A description of a component, that when executed within a simulator, exhibits the behavior of the intended component, such as responding to external events, stimulus, and internal states.

Model – (verb) To create a description of a component in a form that can be connected to other component models and executed within a simulator.

Simulator – (n) A tool that integrates (connects) models for the purpose of executing (running) them, and for observing their interactions, and collecting results. Simulators posses common infrastructure that enables coordinated execution of models, but which is not specific to a particular system being modeled. Such infrastructure includes the synchronization and management of simulated time.

Simulate – (v) To execute models within a simulator. To run simulator.

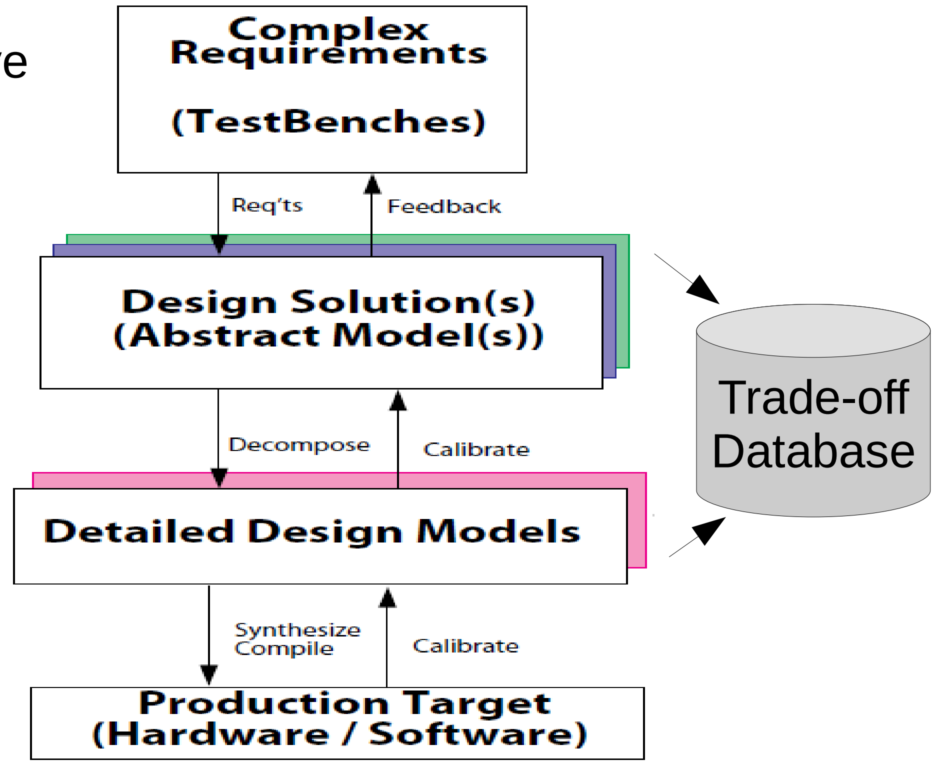
Simulation – (n) A simulator configured with specific models.



How simulations can be used

- Simulator is re-used without modification across many projects by attaching different models
- Simulator enables models to be re-used and integrated to simulate new systems without modifying the models themselves

- **Virtual Prototyping:**
  - Iterative modeling at progressive detail levels reduces design from requirements down to physical system
  - Testing and evaluation occurs throughout process
  - Design is documented as Executable Specification
  - Design alternatives are maintained in trade-off database



Base Model Libraries

Core Models - Hardware/Software Architecture Performance models.
Processor, Bus, Crossbar, Memory, Board & Rack models.
Data Flow Graphs (DFG's), Static&Dynamic Scheduler.
- About 30 models. Some very complex.

General Blocks - BONES-like basic function boxes.
- Includes resource, queuing, and statistics models.
- About 340 models. Most simple, but some are complex.

Human Factors - MicroSaint-like (MAD) work-flow models.
- Includes Task, Queue, Switch, and Resource models.

LAN/WAN - IP-based network models.
- Includes routers, firewalls, switches, hosts, IDS, INE, SONET, hub, subnet, monitor, etc. (VNS like/related)
- About 15 models.

Wireless - Simple radio components models.

Base Model Libraries - continued

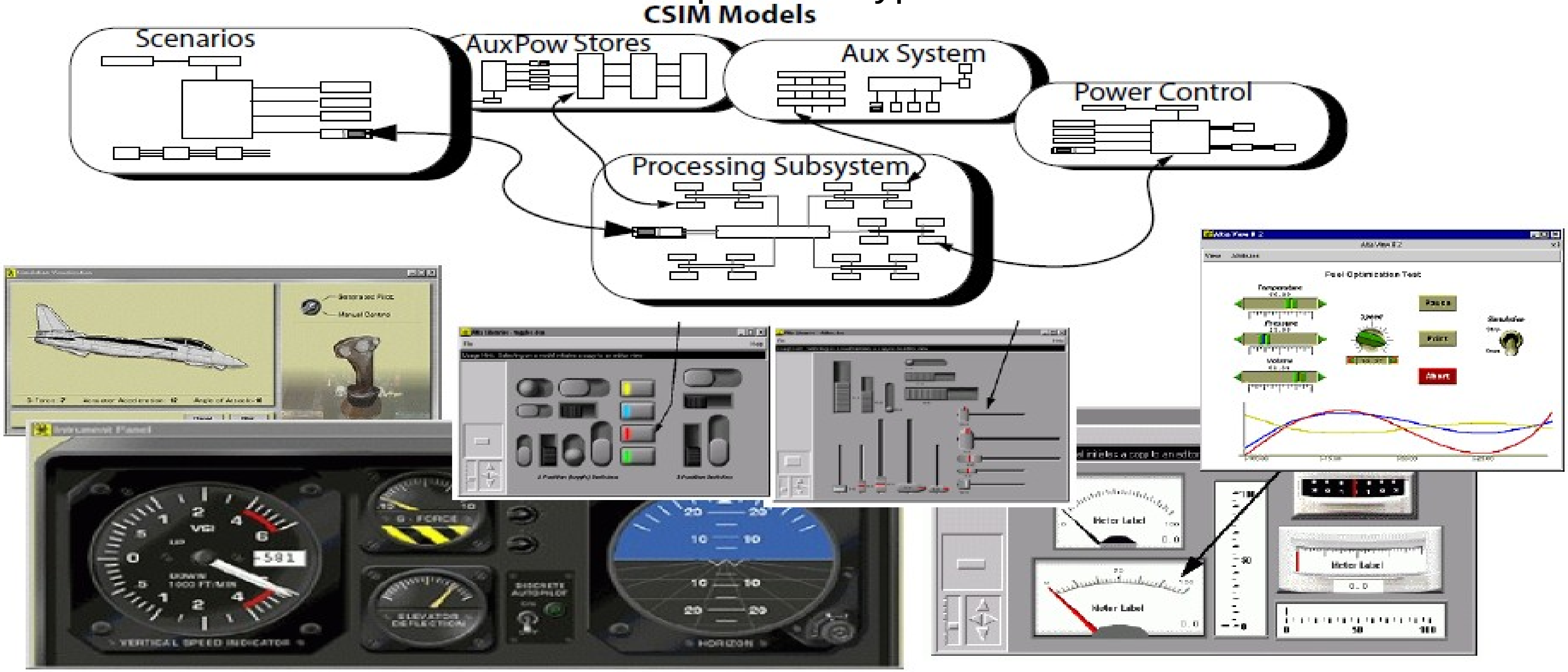
Vehicles Platforms Terrains (VPT) - 3-space models w/movement, trajectories, way-points, mission-plans, spatial-services, maps, vis..
Distributed Simulation Functions - Wormhole model, HLA interface, Multi-Simulator Interface (MS)
User Contributed Models - General purp. functions, slider & button control boxes, generic Pub/Sub messaging functions, etc.

Experimental or Example Libraries: (Not production level)
- **Economic / Social models** - Efficient, scalable to large populations.
- **Complex Vector Math function boxes** - Matrix and vector math func., FFT, image processing, etc..
  - About 50 models.
- **Digital Logic** - Includes basic logic function, and SSI parts, registers, ALUs, general purpose testbench. Includes inertial+transport delays.
  - About 60 models.
- **Analog logic** - Continuous systems w/inertia, frict., energy, momentum.
  - Can model simple analog RLC electrical circuits, thermodynamic systems, mechanical mass/spring/damper systems, fluid reservoir systems
  - Scalable with optimal convergence.

Simulator *In-the-Loop*

- Simulations can be attached to real systems, or ...
- Actual application source code can run as if in actual system.
  - Minimal functional, temporal, or structural distinctions.
  - Enables early and continuous integration and testing.

Virtual Rapid-Prototype



- Developed, validated application software emerges directly from verification models; not separate translation & development process.

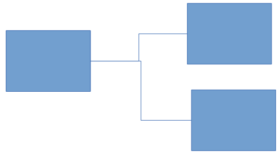
* Graphics from Altia Faceplate

CSIM Model Description

CSIM models consist of two distinct forms of description:

1.) **Architectural or Structural Description:**
– Described graphically (Block diagrams).
- Edited with GUI.
– **Topology** - What connects to what, and how.
– **Model Instantiation** – Sets model-type for each box.

Example:



2.) **Behavioral Description:** (Function + Timing)
– Described textually (C or C++ code).
- Edited with any text editor.
– **Model Type Definitions** - Describes each model-type's Functional & timing behavior

Example:

```
A = B + C;
Delay( 20 mSec );
```

A Simple Example:

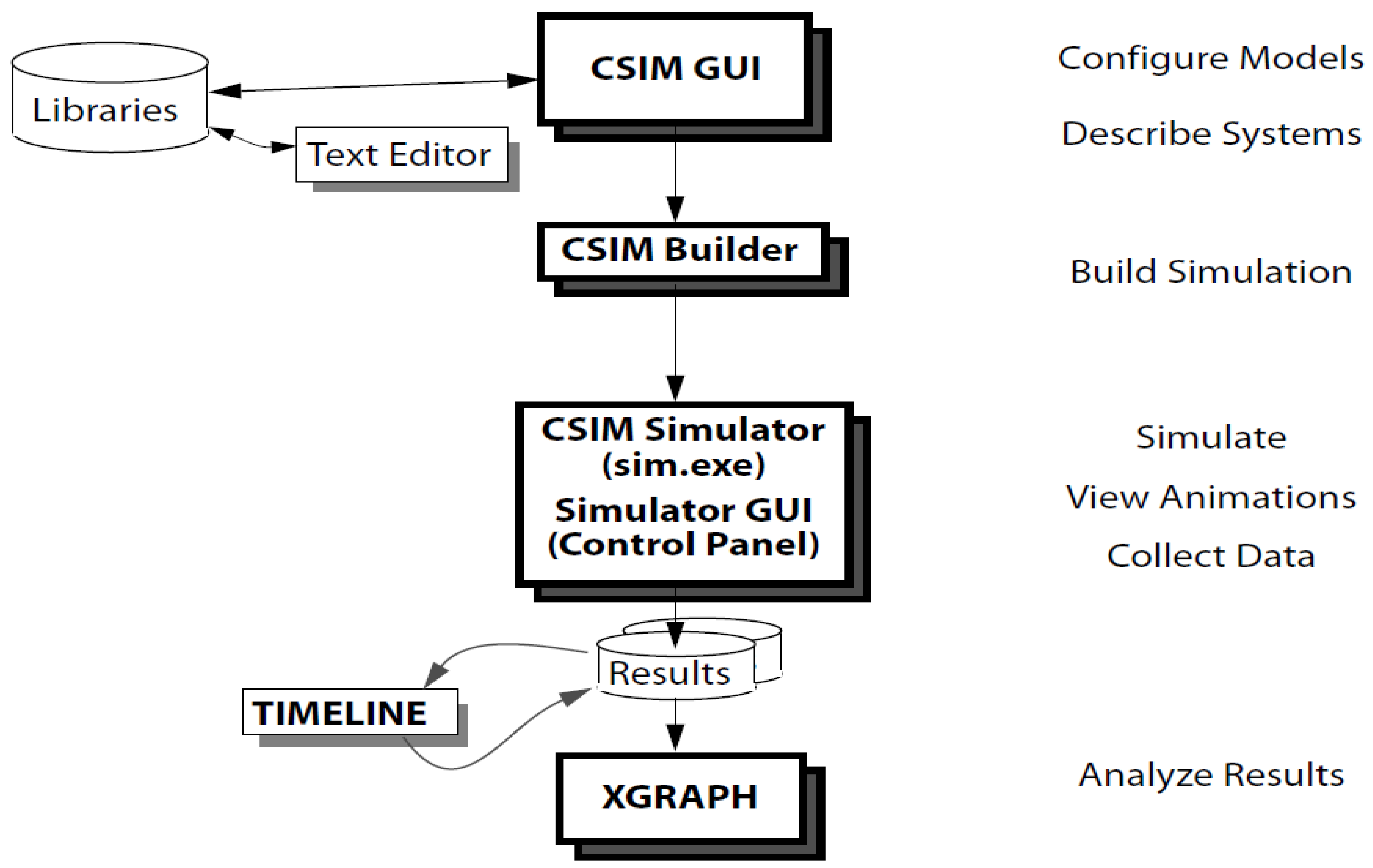
- **Structural description of a system:**

Simple example system.

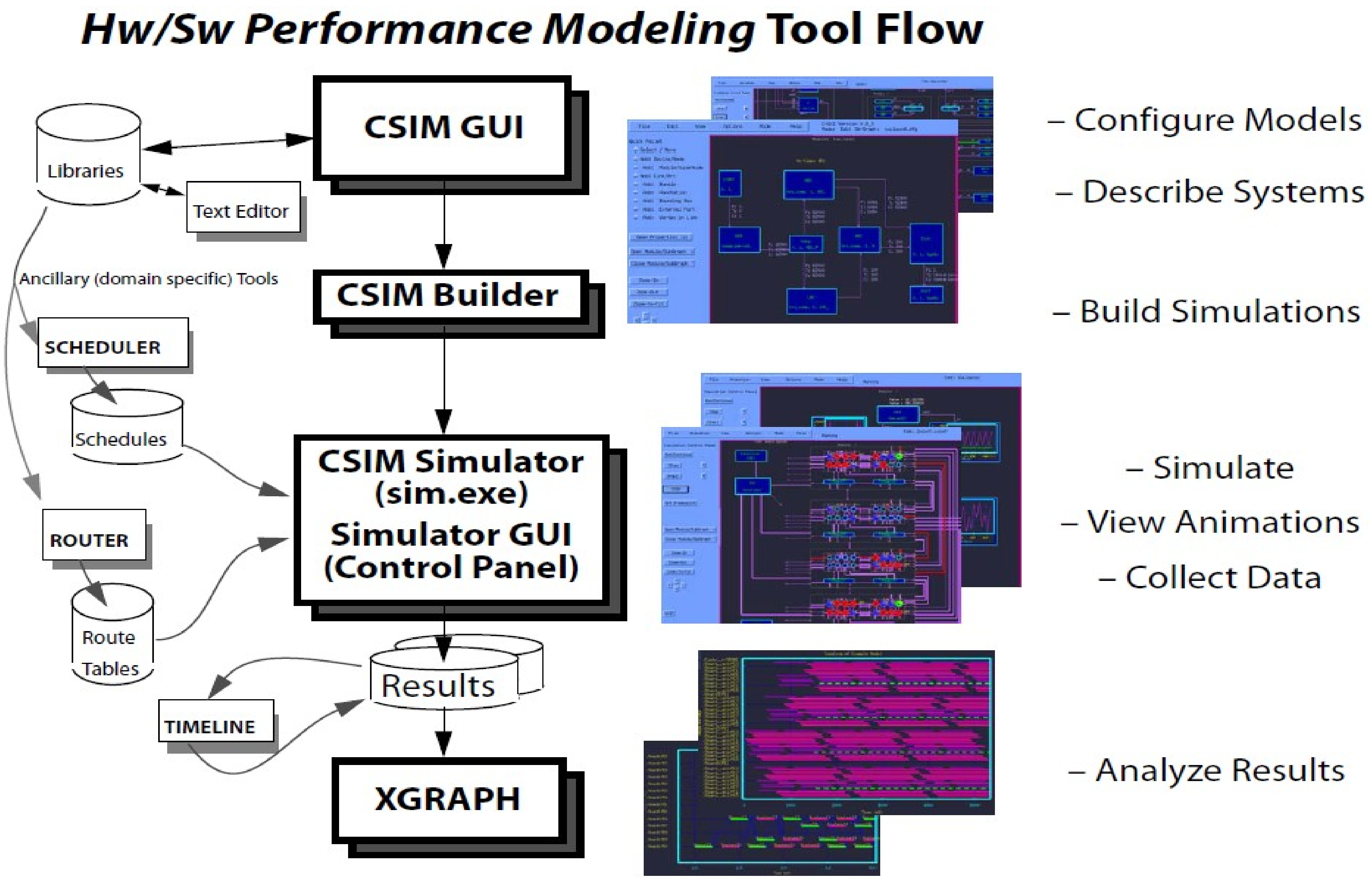
- **Behavioral description of an model-type (box):**

```
DEFINE_DEVICE_TYPE:   MC68030
  PORT_LIST( ioprt );

  /* Local Variables */
  long message, length;

  DEFINE_THREAD: start_up
   {                              /* Any C-code can go between here, */
     message = 1;
     SEND( "ioprt", message, 1);
     RECEIVE( "ioprt", &message, &length );
     printf(" Received reply.\n");
    }                             /* ... and here. */
  END_DEFINE_THREAD.

END_DEFINE_DEVICE_TYPE.
```

Basic Tool Flow

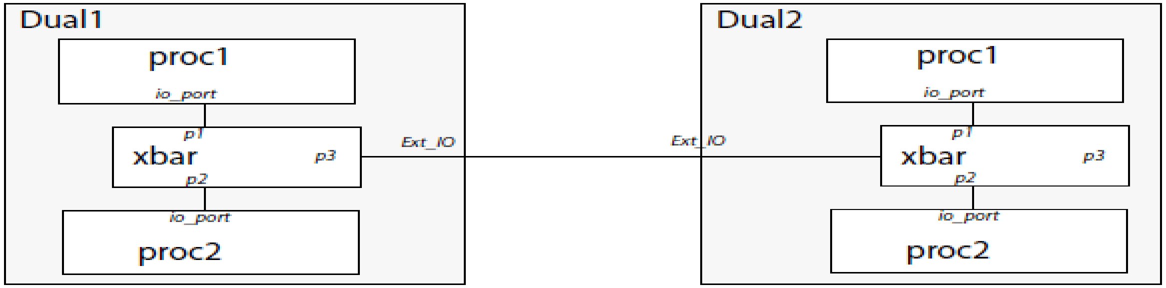


Other Tool Flows



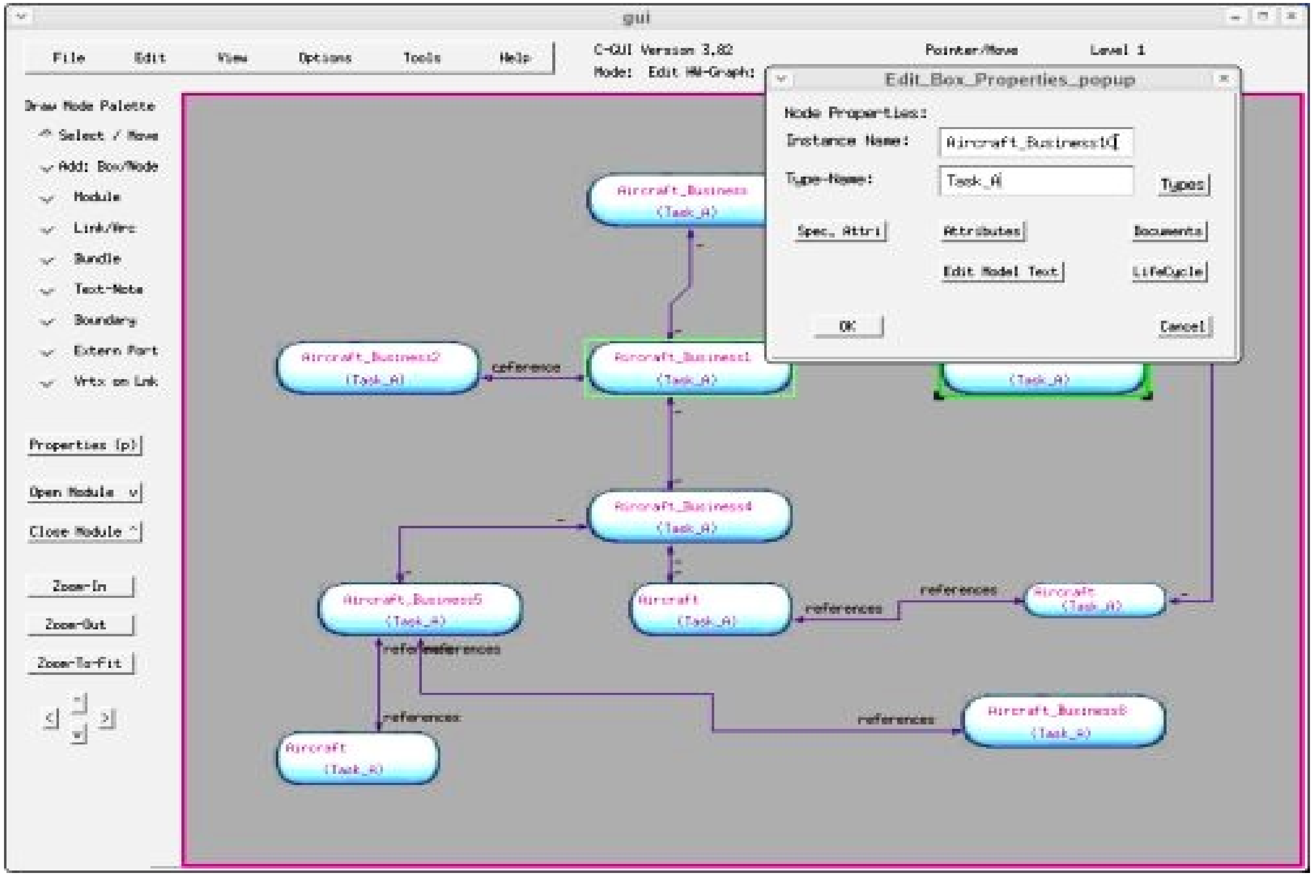
Structural Definition

- A structural block diagram describes the topology of a module.
- A module diagram contains *boxes* connected by *links*.
  - *Boxes* represent entities or sub-modules.
    - Leaf-entities represent behavioral nodes (contain C-code).
    - Sub-modules produce hierarchy by referencing lower diagrams.
  - *Links* represent connections between boxes.
- Module-boxes are drawn with thicker lines than leaf-entity-boxes.
- Double-clicking, or opening, a module-box, opens sub-module diagram.
- Attributes:
  - Boxes: Instance Name, Type.
  - Links: Direction (smplx, hdplx, fdplx) (* = full-duplex),
Transfer Rate (* = infinity MB/S),
Fixed Overhead (* = 0.0 uS),
Queue/Buffer Size (* = Messages/-Bytes).
- The highest module in the hierarchy is called top_level.



Graphical User Interface

- GUI provides convenient entry/editing of structure diagrams.
- Buttons for building & running simulations.
- Controls for running & visualizing simulations.



Box Behavior Description

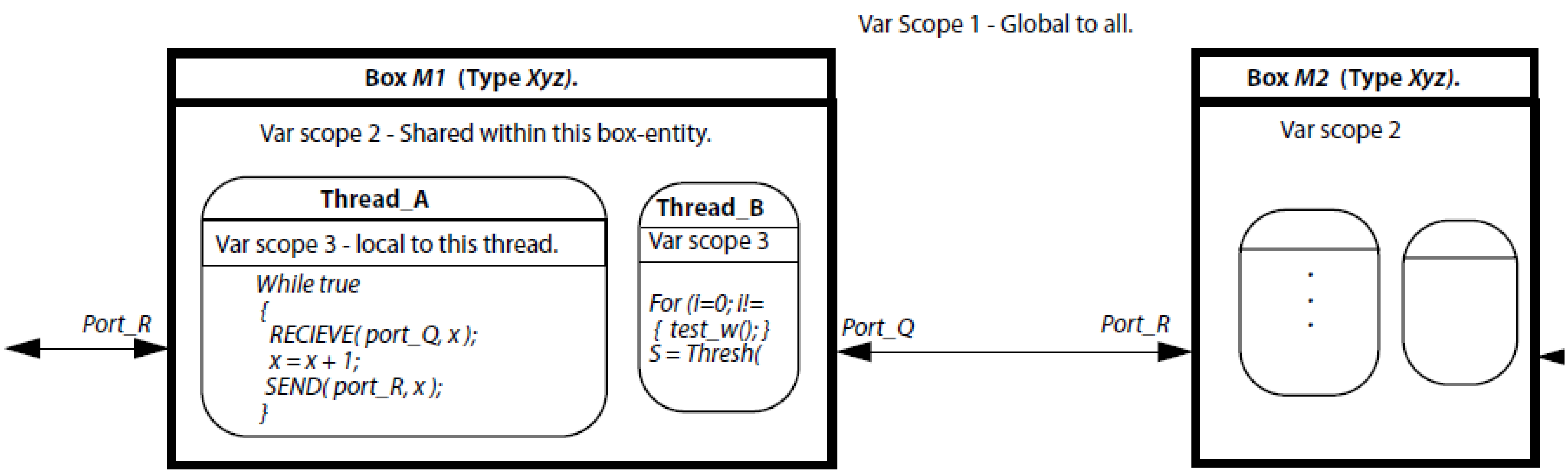
Threads and Variable Scoping

Thread – Software process, shares variables with other threads in a box (entity).

- All boxes must have a thread called start_up.
- The start_up thread is started in each entity at the start of a simulation.
- Threads trigger the activation of other threads, created/ended dynamically.
- Multiple threads can be active concurrently within each entity.

Variable Scoping Levels - Three distinct levels of variable scoping:

1. Global - Globally accessible to all entity boxes and all threads within them.
2. Shared - Shared by all threads within an entity. Local to each entity instance.
3. Local - Local to each thread instance in each entity.



Box Behavior Definition

The behavior of each model-type must be defined.

```
DEFINE_DEVICE_TYPE:   MC68030
  /* Shared Variables */
  long message;

  DEFINE_THREAD:  start_up
  {
   if (strcmp(MY_NAME,"/Kingpin")==0)
    {
     message = 1;
     SEND( "port1", (void *)message, 1.0);
     TRIGGER_THREAD( gen, 100.0, THREAD_VAR );
    }
   }
  END_DEFINE_THREAD.

  DEFINE_THREAD:  gen
   {
   message = message + 1;
   SEND( "port1", message, 1.0 );
   TRIGGER_THREAD( gen, 100.5, 0 );
   DELAY( (double)message + 10.0 );
   TRIGGER_THREAD( gen, 20.0, 0 );
   }
  END_DEFINE_THREAD.

END_DEFINE_DEVICE_TYPE.
```

Keyword to begin definition
Name of entity type.
Shared variables to each instance
of this model type.
Keyword to begin thread.
Name of thread.

Code Thread.
(Any C-code, plus CSIM extensions.)

Keyword to end thread.

Code Thread.
(Any C-code, plus CSIM extensions.)

Keyword to end definition

Behavior Definition

Data type definitions, data structures, global variables, C-macros, and common subroutines are defined in common area.

```
DEFINE_GLOBAL:

    int population;
    float clock_frequency=80e6;

    int scale_vector( x, y )
    { /* C-subroutine code*/
        ...
    }

END_DEFINE_GLOBAL.
```

These objects can be accessed from any of the threads.

#includes should also be placed in these sections.

Anything outside Define_... blocks is ignored !!!

Model Behavior Definition

C Construct Extensions for Behavior Descriptions:

Standard predefined variables:

MY_NAME - Entity instance name (char string).
CSIM_TIME - Double floating-point value current simulation time (ex. Seconds).
THREAD_VAR - Pointer to thread-unique variables.

Predefined Functions:

DELAY(*delay_amt* ***)*** - Causes thread to sleep for specified duration relative to current time.

TRIGGER_THREAD(*thread, delay_amt, thread_var* ***)***
- Spawns the start of the specified thread within entity after delay_amt.

SEND(*port, message_ptr, length* ***)*** - Causes message-data to be sent out specified port to another entity.

RECEIVE(*port, &message, &length* ***)*** - If incoming message has arrived and is waiting on the specified port's queue, then it will be dequeued and returned in the message variable. Otherwise, the thread will block (sleep) until the requested data length arrives.

CHECK(*port, &status* ***)*** - Checks for pending messages without blocking or dequeuing.

Modeling Constructs

- **Including files:** GUI: File / Import / By-Reference

  Inserts:

  %include

  - Works like the regular C #include, but expanded by CSIM preprocessor.
  - Generated by Import-by-Reference in GUI.

  Example:

  %include subroutines.sim

- **Halting simulation from within model:**

  halt();

  - Causes the simulation to pause, returns control to user.
  - User can resume or quit.

  Example:

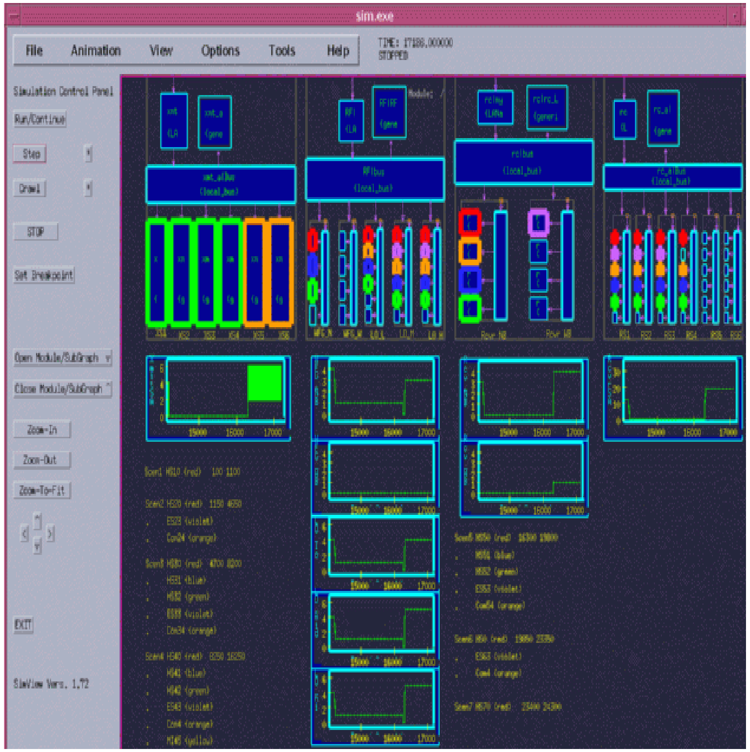
  ```
  if (unexpected_event)
  halt();
  ```

  Useful for:

  - Inserting conditional breakpoints into a simulation.
  - Breaking in (or on) specific subroutines or lines of code.
  - Calling attention when assertions have been violated.
  - Stepping by significant events instead of time amounts.
  - Detecting and forcing a simulation end-point.

See *CSIM_HALT_POPUP(“Message”);* - for graphical sims.

Animation Control



User Customizable Animations:

Box, Link Colors:

- Boxes and links change colors from user’s code during simulation.

highlight_box(color);
highlight_link(port_name, color);

Annotations:

- Annotate writes textual information on the graphical display near boxes during simulations.

Annotate(char_string, color, xoffset, yoffset);
Annotate(“Transmitting Pulse 5A”, Green, 0.0, -0.5);

- Use to view changing state information or to note specific or unusual events.
- More visual than prints because it is placed with respective box in diagram.

Supports Third-party Visualizations - Altia Faceplate, WinFrame-3D, Otk, etc.

Pop-up Messages From Models

– A model can popup a message window. Call the following:

*CSIM_HALT_POPUP(char *message)*

Your message as the argument.

– A popup window appears with your message in it.
– Popup has a OK button on it.

Example:

```
CSIM_HALT_POPUP( "Special event occurred." );
```

Or,

```
// Copy your message into a string ...
char mymssg[60];
sprintf(mymssg,"%s Over temperature by %f degrees", MY_NAME, x);
CSIM_HALT_POPUP( mymssg );
```

More Modeling Constructs

- **WAIT and RESUME:** (Basis of wireless models)
  - Controls thread execution, or synchronizes threads to events.
  - Convenient for implementing general resource models.
  - *WAIT* causes thread to sleep until awoken by *RESUME* in another thread.
  - *WAIT* and *RESUME* operate on *SYNCHRON* variables.
  - *WAIT*s and *RESUME*s can be queued or not-queued.

Example:

```
DEFINE_DEVICE_TYPE: Model_XYZ
  SYNCHRON *synchpt_A;

  DEFINE_THREAD: start_up
   {
        synchpt_A = NEW_SYNCHRON();
        TRIGGER_THREAD( processB, 0.0, 0 );
        DELAY( 100.0 );
        RESUME( &synchpt_A, NONQUEUABLE );     /* NOTE the & */
   }
  END_DEFINE_THREAD.

  DEFINE_THREAD: processB
   {
        WAIT( &synchpt_A, QUEUABLE );            /* NOTE the & */
        csim_printf("Process_B awake\n");
   }
  END_DEFINE_THREAD.

END_DEFINE_DEVICE_TYPE.
```

Instance Attributes

- Specify distinct arbitrary attributes for each box instance.
- Attributes can be specified at any level of the hierarchy in GUI.
- Assign attributes to a model-box when editing it's properties.
- Attributes work like macros. One attribute equation per line.

attribute_name = value/expression

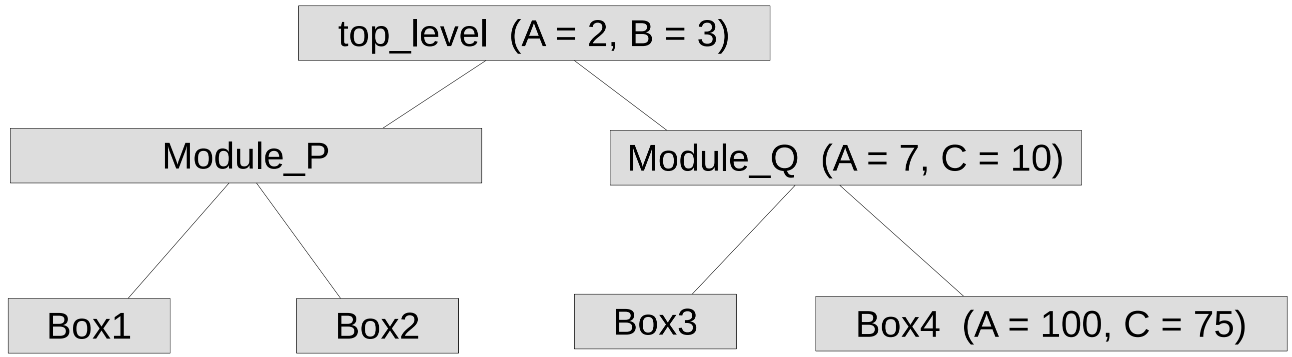
Example:

Xproduce = 5
*Yconsume = Xrate * 50.1*

- Access attribute values within models: *CSIM_GET_ATTRIBUTE()*

*CSIM_GET_ATTRIBUTE(char *attribute_name, char *value, int maxstrlen)*

- Attributes inherit downward through graph hierarchy.
  Can be specified at any level.



Instance Attributes - continued

Model code within the boxes (on previous slide) sees the following attribute values:

Module_P/Box1: A = 2, B = 3

Module_P/Box2: A = 2, B = 3

Module_Q/Box3: A = 7, B = 3, C = 10

Module_Q/Box4: A = 100, B = 3, C = 10, D = 75

CSIM Installation

Platform	Method
- Linux (Redhat, Ubuntu, ...)	1. *tar xfz csim_vxx.tgz* 2. *sh csim_xx/install_csim_posix.sh* 3. start by double-clicking csim desktop icon
- MS Windows	1. *unzip csim_vxx.zip* 2. Run *mswin_installer_gui* 3. Drag start-icon to desktop
- Mac OSx	1. Install Oracle VirtualBox 2. Import *csim_xx.ova* 3. Start VM, double-click csim icon

* The VirtualBox method can also be used on the other platforms including Linux, Microsoft, and Sun Solaris.

* Detailed install instructions are included with each release package.

CSIM Package Directories

```
$CSIM_ROOT
   ├── tools                      <-- (Main tool directory)
   │     └── general_utilities
   ├── model_libs
   │     ├── core_models          ┐
   │     ├── general_blocks       │
   │     ├── lan_lib              ├ (base component models)
   │     ├── digilogic            │
   │     └── user_contributed     ┘
   └── demo_examples
         ├── demo0
         │   ...
         └── demo7
```

- Always source *$CSIM_ROOT/setup* prior to all sessions.
- Then you may want to *export CSIM_GUI_SETUPS* to your own customized *gui_setup* file. (*export* is Linux bash syntax)
- Set text editor: *export CSIM_TEXT_EDITOR myeditor*.
- You work with your models in your own (arbitrary) directories while referencing tools and models under $CSIM_ROOT as needed.

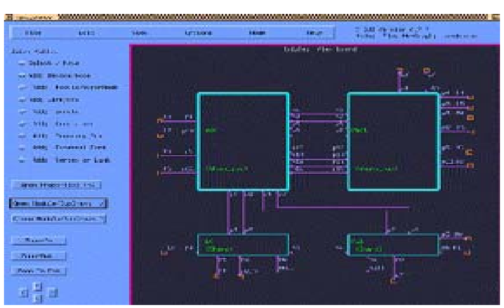
Tool Usage

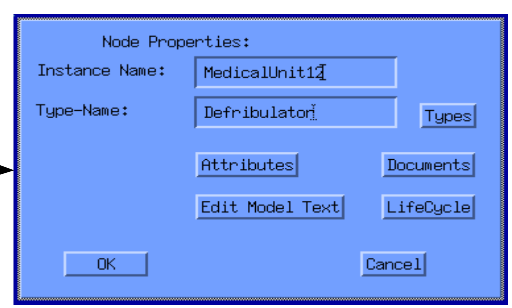
- **GUI** - Capturing/editing design diagrams
- **CSIM** PreProcessor - Building simulations
- **ROUTER** - Building routing tables
- **SCHEDULER** - Generating application software
- **SIMview** - Running simulations
- **XGRAPH** - Viewing results
- **TIMELINE** Postprocessor - Customizing activity timeline plots
- **CONTENTION_VIEWER** - Viewing network contentions
- **C-2-HTML** - Auto-documentation of model code for understanding.
- **ITERATOR** - Multiple simulation launcher / aggregater.
- **ScenGen** - Scenario entry tool and generator.
- **WinFrame-3D** (WF3D) - Animation Viewer

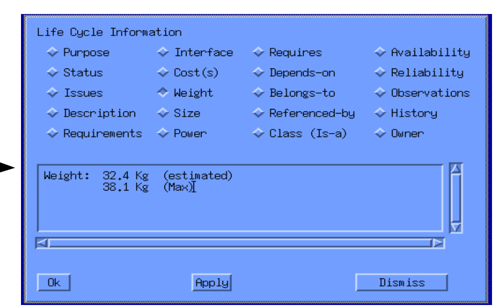
CSIM - GUI

GUI

- CSIM's main graphical interface & diagram editor.
- Preferences file: *export GUI_SETUP_FILE*
  (defaults to: *$CSIM/tools/$CSIM_MTYPE/gui_setups*)
  - Sticky vs. non-sticky draw mode. – Snap/Gravity settings.
  - Background colors and box styles. – Grid on/off/size setting.
  - Initial window size, etc.. – Display settings.
  - Preferred Text Editor, Printer, etc.. – Journaling period.
- Hierarchy-by-Reference - Diagrams, Modules-Boxes, Bundles-Links.
  - Double-clicking module opens sub-diagram.
  - Sub-diagram is common to all instance of given module type!
- Accessing Object-Attributes, Graph-Parameters, Macros, or Variables.
  - All are in common name-space. Can reference one another.
  - Edit global attributes under Edit/Macro or Edit/Variables.
  - Edit object attributes under selected object’s Properties popup.

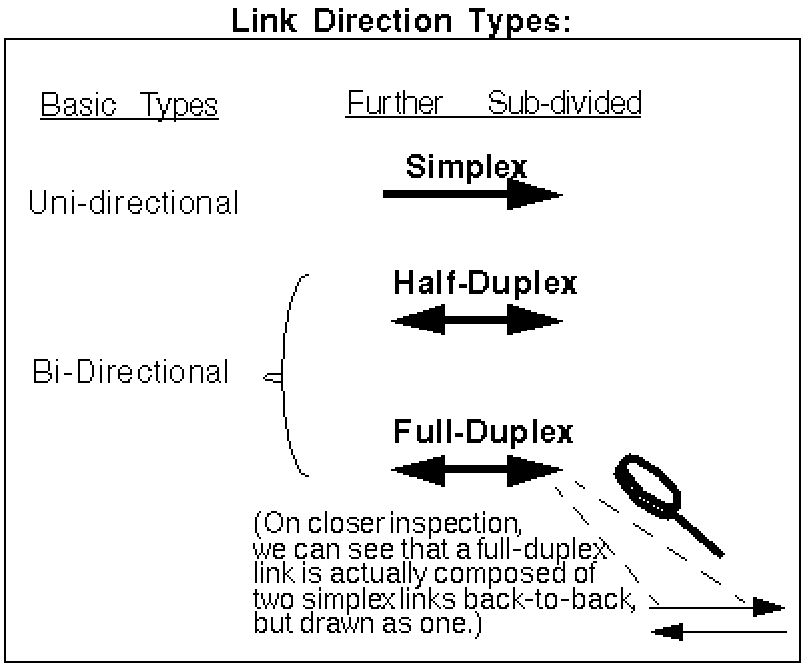


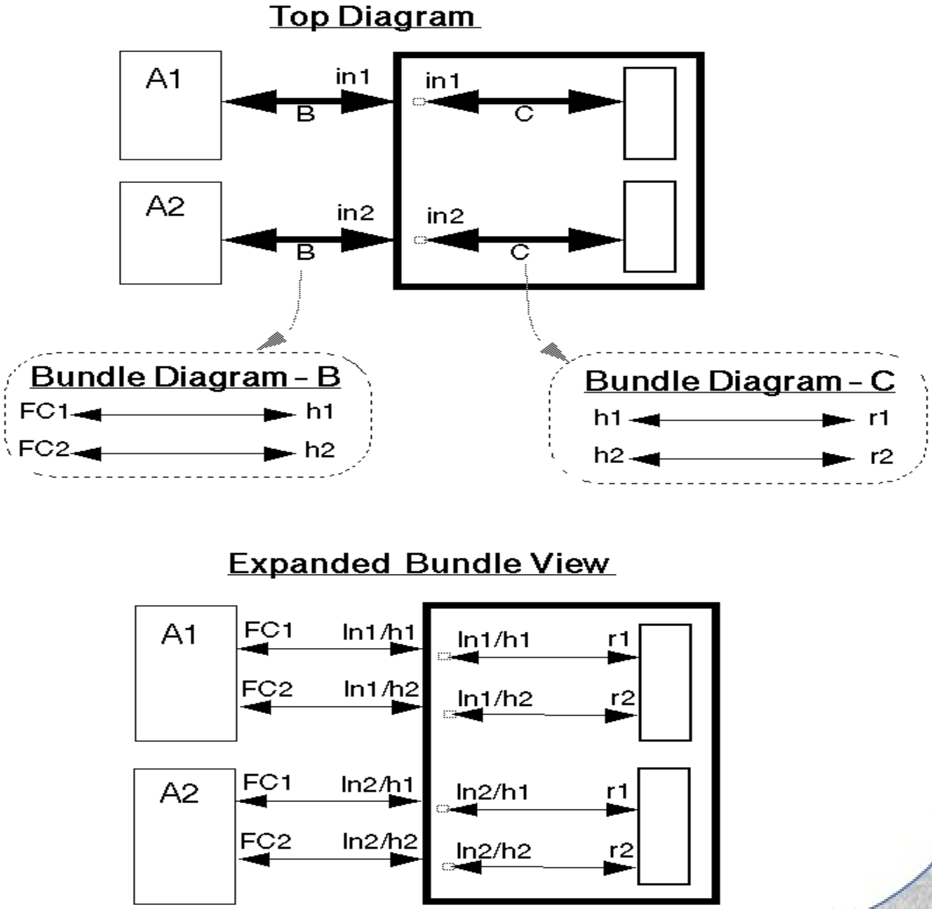




CSIM - GUI (continued)

- Export diagrams directly to Office tools. (*File / Print / To File / Office)*.
- Save image files for documents. (*File / Print / To File / Image*).
- Link directions, simplex = uni-directional, duplex = bi-directional
- Bundles: Sub-diagrams for wire-groups. Like Module is to Box.
  - Port-names become concatenated across levels. Must be Unique.



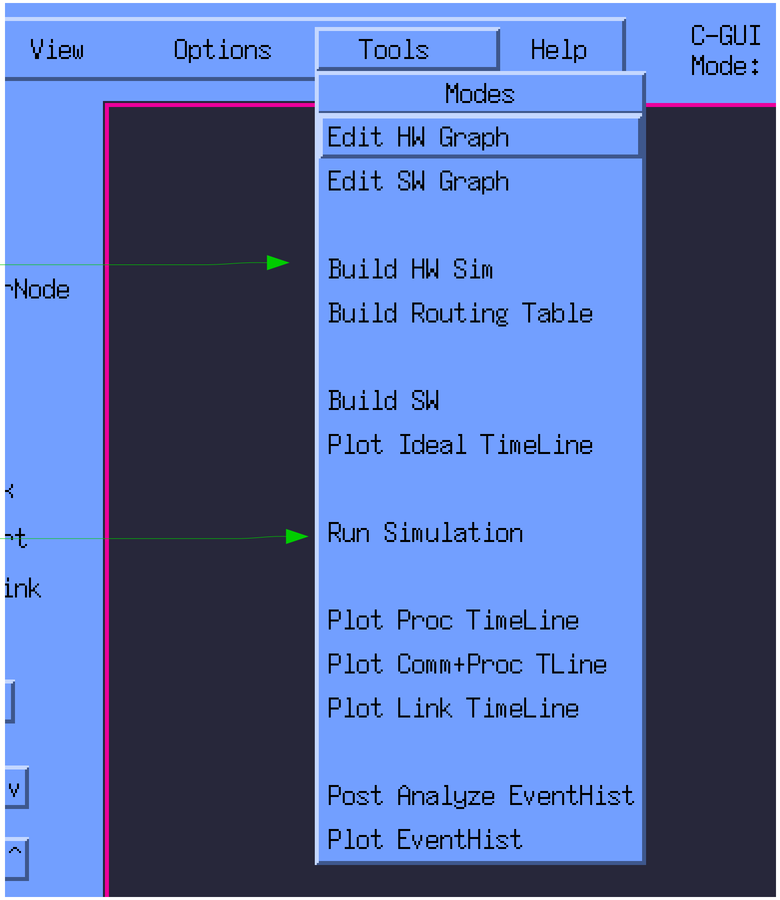


Building Simulations

1. Define system architecture structure and model behaviors.
   – These may be placed in a single or separate files via file/import menu.

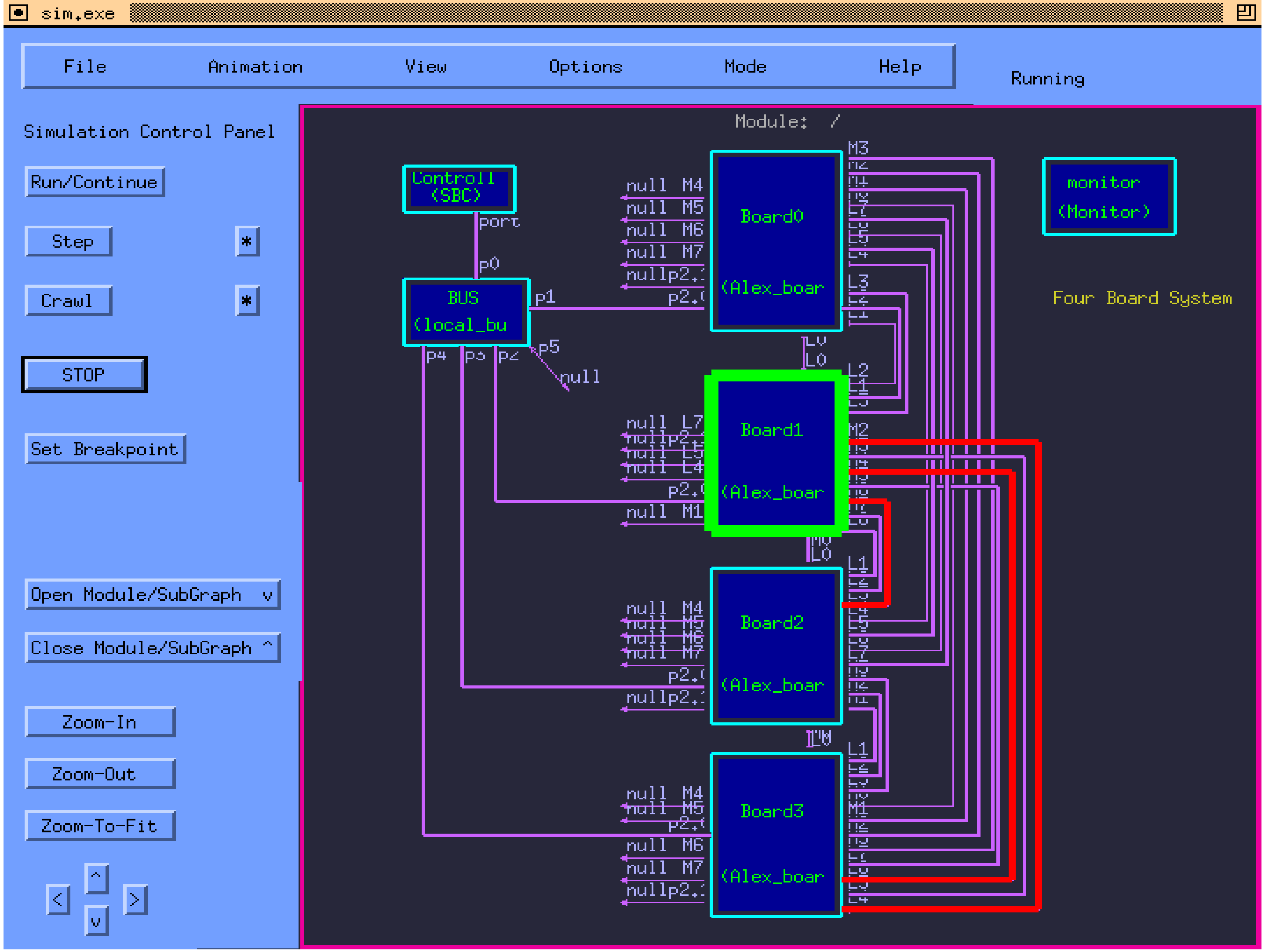
2. Run CSIM preprocessor from *Tools* menu.
   – Analyzes and processes the files, and links CSIM simulation kernel to produce executable simulation file.

3. Run simulation by selecting *Run* under the Tools menu or by calling *sim.exe*.
   – Two simulation modes are available:
   - Text Mode - Good for batch script driven, automatic overnight runs.
   - Graphical Control Panel - Good for animations. Easy to learn and use.



Running Simulations

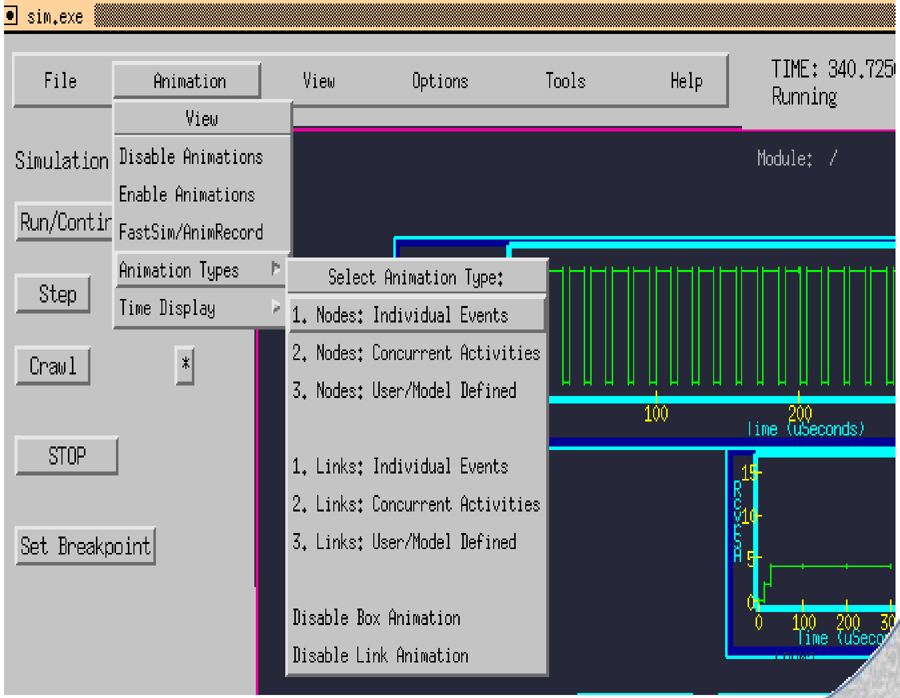
- Interactive simulations can be run from SimView control panel.



SimView - Simulation Control Panel

SimView -

- Run control - Run, Crawl, Step, Stop, Breakpoints.
- Navigate diagrams as in the main CSIM-GUI.
- Time & Status Displays.
- Animation Options:
  - Animation On/Off
  - Time Display Updating
  - Animation types, links/boxes, concurrency, built-in/user.
- View Options:
  - Port names
  - Aspect Ratio Lock
  - Flatten hierarchies
- Option Menu:
  - Scrolling verbosities
  - Examine model state variables
  - Examine link states
  - List event queue.
- Tools:
  - View Time Activity Plots.
  - Print diagrams/images.



Text-Mode Simulation

- Alternative to SimView graphical simulation. Great for batch jobs!
- Build with: *csim -nongraphical arch.sim* (or *Tools/Modify* GUI menu.)
- Run simulation by invoking *sim.exe*
- When simulation comes up, you will see the simulation prompt.
- At the prompt, you can type *h* for help at any time.
- It is common to set simulation displays and break-points at this time.
- You can initiate and control the simulation by *run, step,* or *crawl*.

  Example:

  sim> s

  (step amount = 1.0, stepping by 1.0)

  or

  sim> step 0.25

  (step amount = 0.25, stepping by 0.25)

  sim> s

  (step amount = 0.25, stepping by 0.25)
- Crawl can be used to sequence by single events.
- *crawl xx* - sequences to event-xx within the current time.
- *run* - Starts simulation running until breakpoint or end, also *r*.
- *quit* - Exits from simulation, also q.
- *sim_status* - tells what device is being serviced, what event is being processed, and what event the simulator is on within the time instant.

Running Simulations

Summary of run-time commands:

verbosity settings, (v), under view menu.
run - Run or resume, button.
step - Step by time amount (s or step xx), button.
crawl - Step by a event(s), (c or c xx), button.
break-points - Set break-points, (b), button.
show_links - Show status of all links.
examine_link - Examine status and queue of specific link.
stats - Send link utilization statistics to file.
sim_status - Tells what is about be executed, device, event, delta-cycle, etc..
show_active_links - Show just the active links.
show_event_queue - Show the current event queue.
fshow_links - Dump links status snap-shot to a file.
list_devices - Show a list of all the devices names in the simulated system.
list_variables - Show a list of the shared variables contained by a box and their current values.
time - Print the current simulation time.
q = exit

The Simulation Cycle

Time Advance Loop

Advance time to next scheduled event.

While there are processes scheduled to execute at this time.

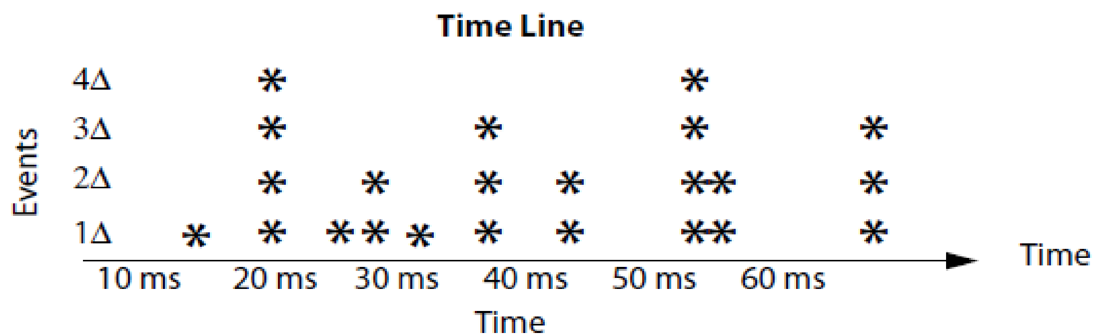
Delta Cycle Loop

Execute any processes scheduled for this time

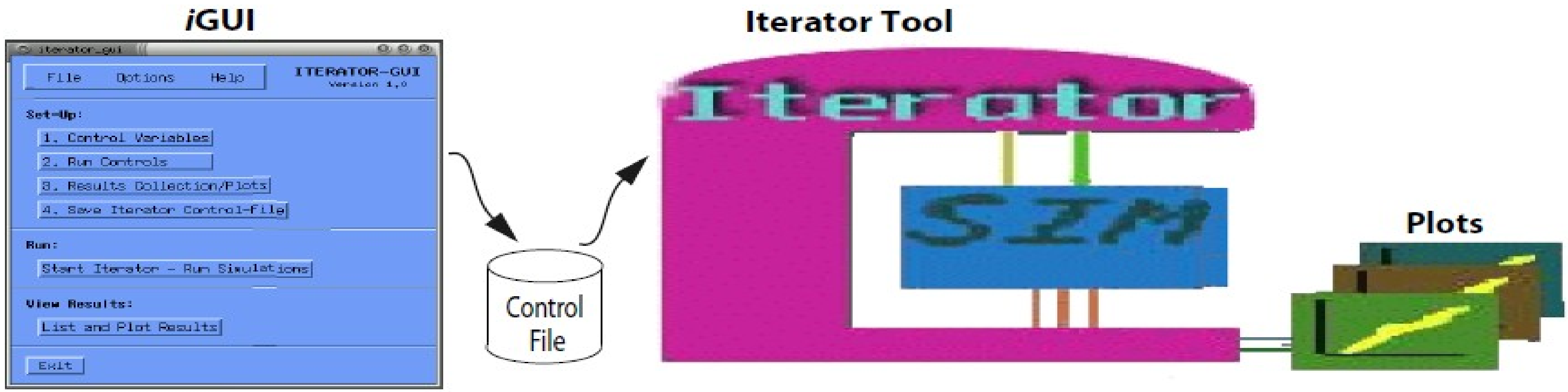
(
a.) Schedule new events and processes,
b.) Assign new values to variables.
c.) Record new values to be assigned to ports.
)

Advance schedule to new events.

- Delta represents sequence only (infinitesimal delay), without time increment.



Iterator Tool & iGUI



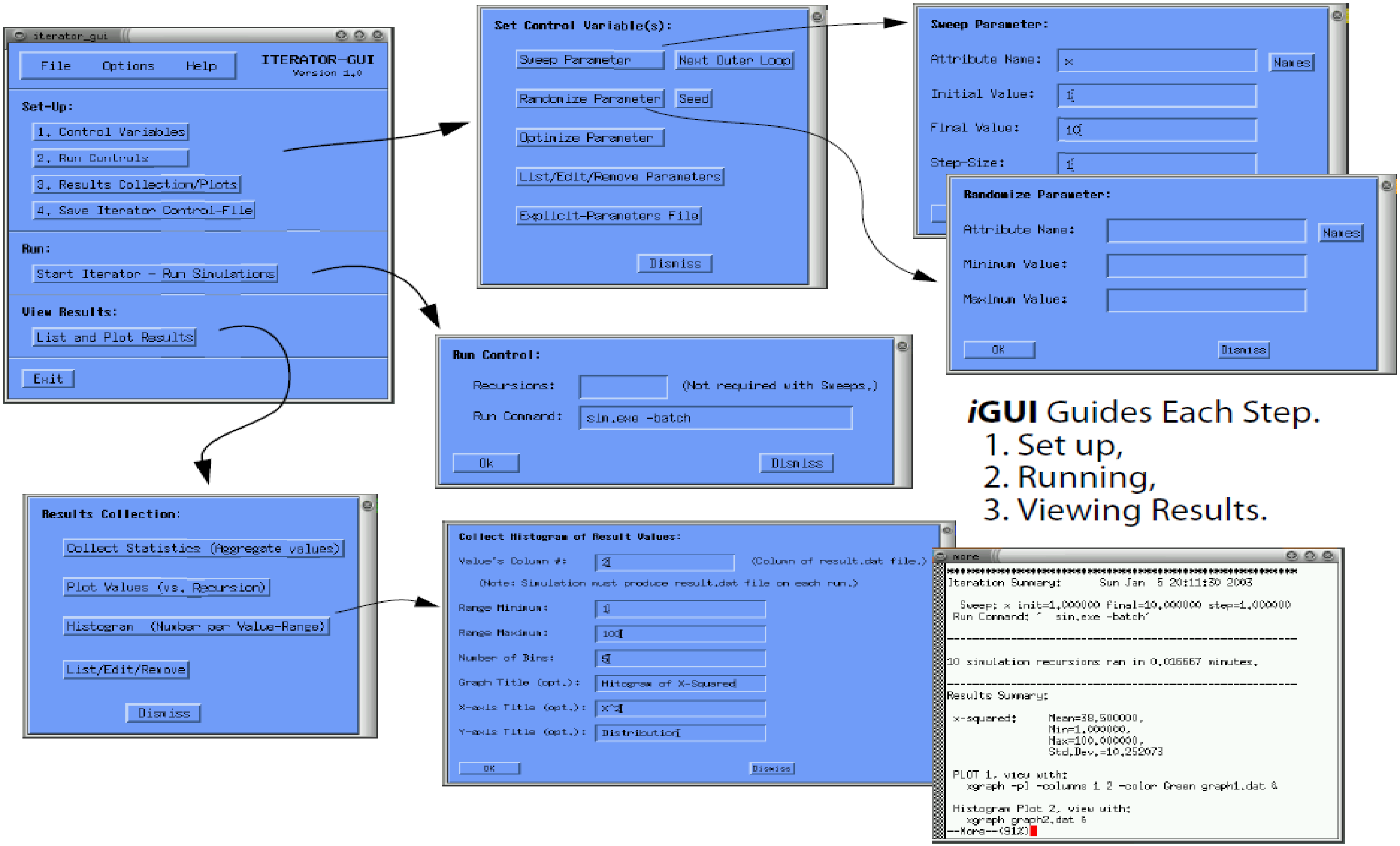
<u>Iterator:</u>

* Launch multiple simulations,
* Sweep parameters over specified ranges and step sizes for recursions, or,
* Vary parameters randomly over ranges - Monte Carlo simulation,
* Change parameters without re-building/re-compiling.
* Aggregate the results of multiple simulations for listing and/or plotting.
* Plot histograms.
* Optimize control variables.

<u>Iterator-GUI (*i*GUI)</u>

I. Set-up parameters/ranges and statistics to be gathered,
II. Launch recursions/simulation(s),
III. View results from the multiple runs.

Iterator Tool & iGUI (continued)

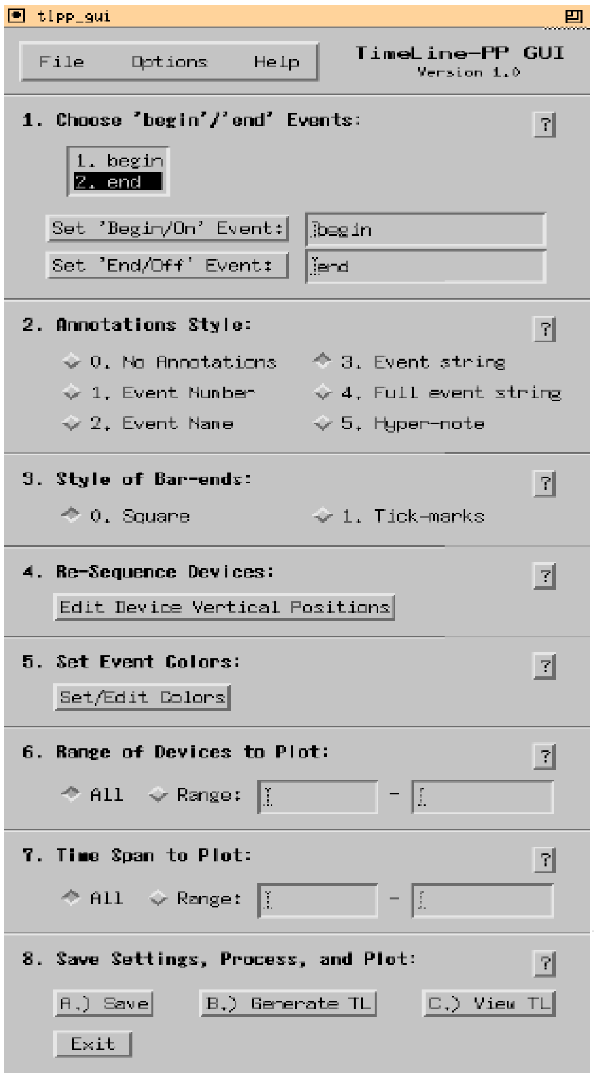


*i*GUI Guides Each Step.
1. Set up,
2. Running,
3. Viewing Results.

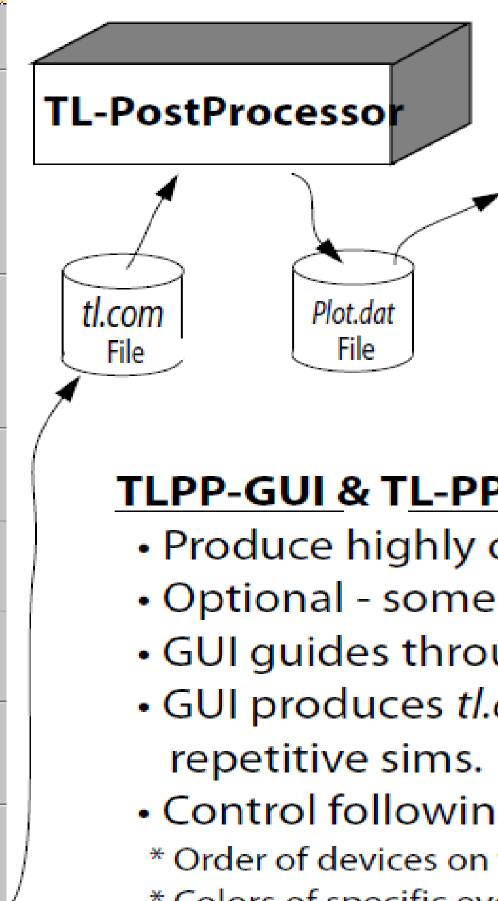
Time-Line Tool & GUI

CSIM

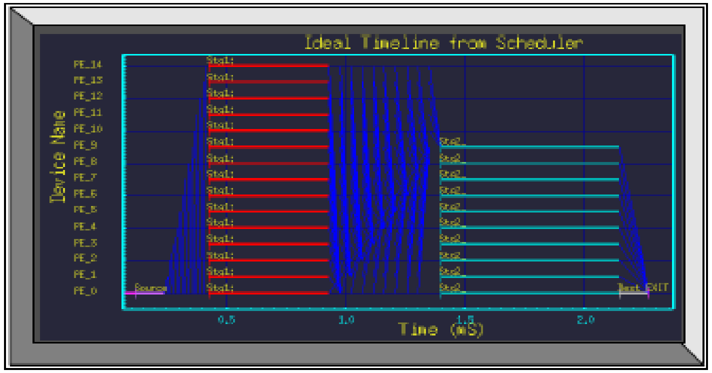
Time-Line GUI



Time-Line-PP



XGRAPH



TLPP-GUI & TL-PP:

- Produce highly customized time-line graphs.
- Optional - some models produce plots directly.
- GUI guides through each step.
- GUI produces *tl.com* file. Can be re-used for repetitive sims.
- Control following aspects:
  - * Order of devices on the y-axis,
  - * Colors of specific event-types,
  - * Annotation styles,
  - * Sub-sets of devices displayed,
  - * Time ranges displayed
  - * Graph titles.
  - * Styles and thickness of time-line bars,

Models & Modeling

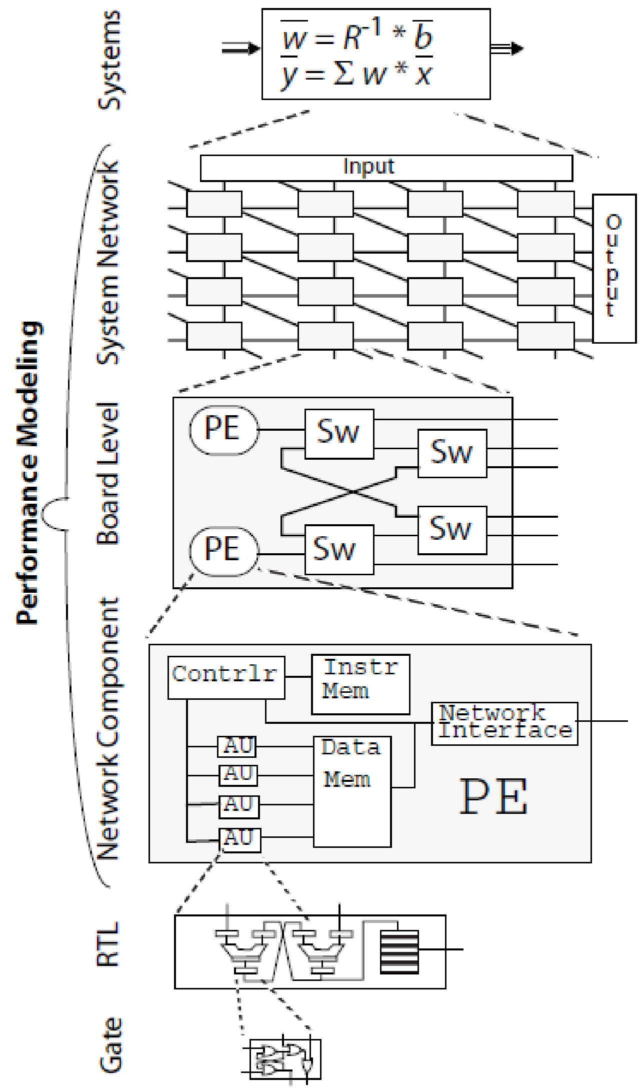
– Techniques and Conventions
– CSIM language features and extensions (general)
– Performance modeling concepts
– Performance modeling library - modeling hardware architecture
– Data Flow Graphs - describing application software
– Model Calibration, Validation & Verification

Performance Model Libraries

- Performance Model => Time-related aspects,
= Structure + Timing only,
(almost no Functional detail)

- A set of special purpose utilities support the performance models.

- The performance models are intended to investigate system architecture related issues, such as:
  - Network topologies,
  - Bottlenecks
  - Scenario-to-network task mappings,
  - Resource utilization
  - Time-lines of proposed systems.
  - Determine total system processing throughputs and latencies.

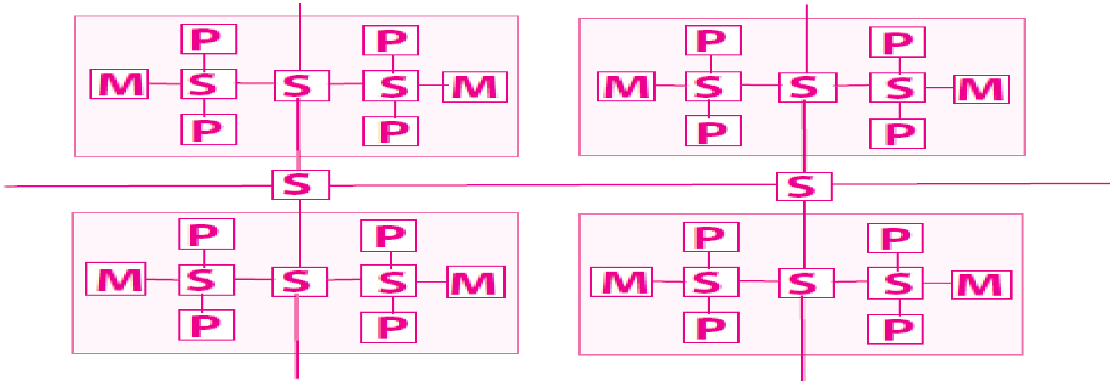


Hw/Sw-Architecture Paradigm

(Computer Hardware / Software modeling)
(Multi-core, Multi-processor, and/or multi-process systems)

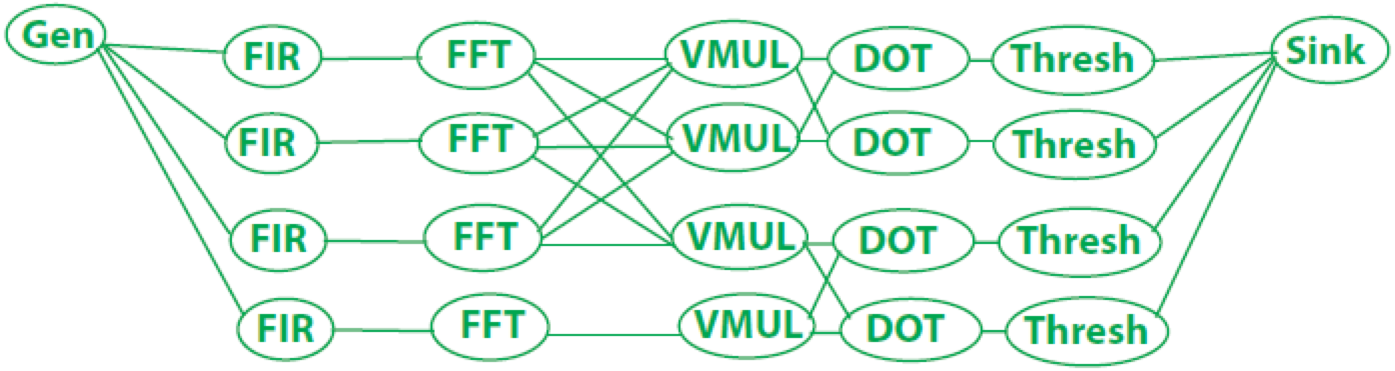
Network Hardware Architecture:

Processor, Memory, and Switch Nodes Connected by Network Links

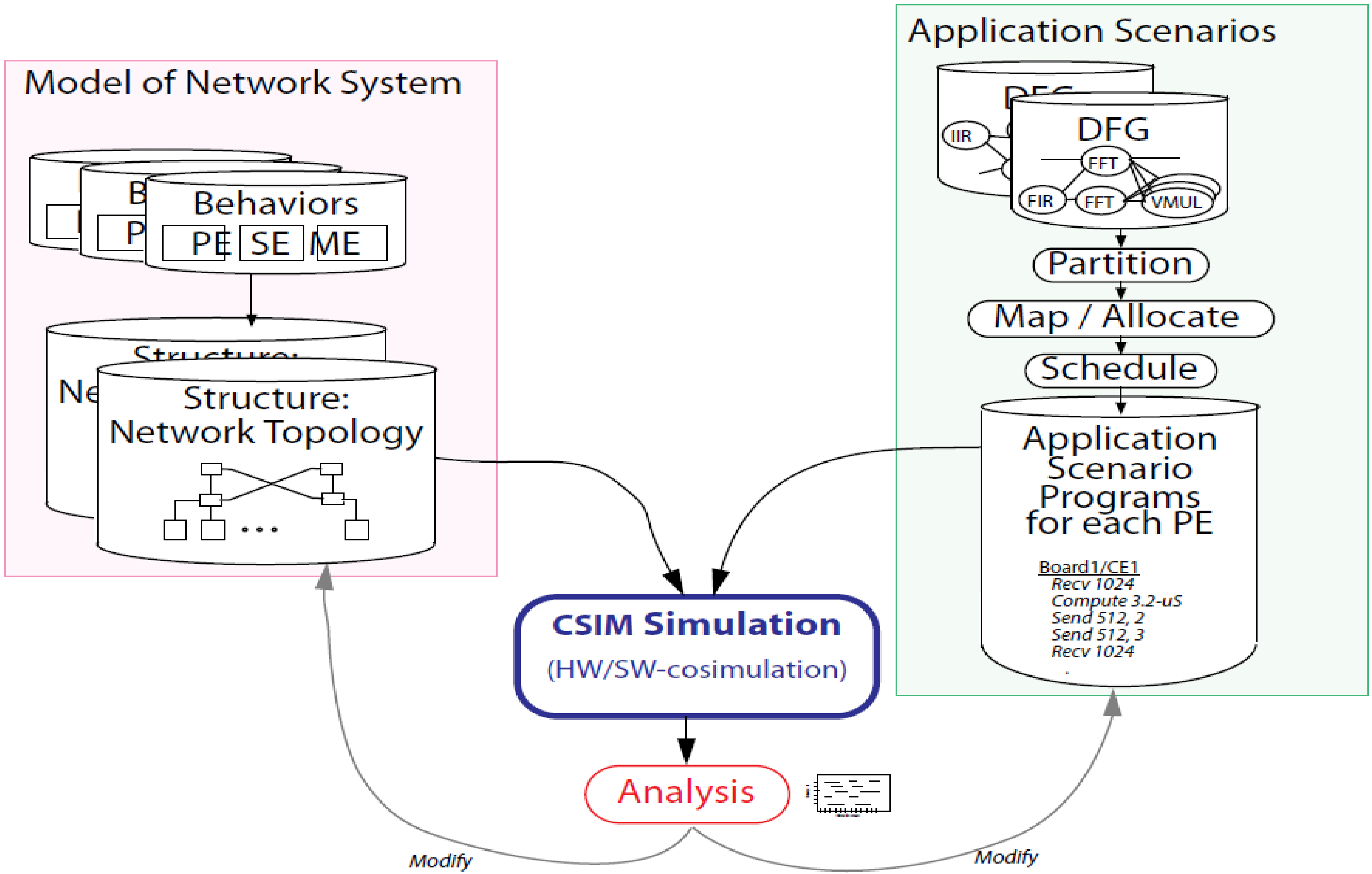


Software Application Flow Diagram:

Primitive Tasks and their Data Dependencies as Data Flow Graph (DFG)

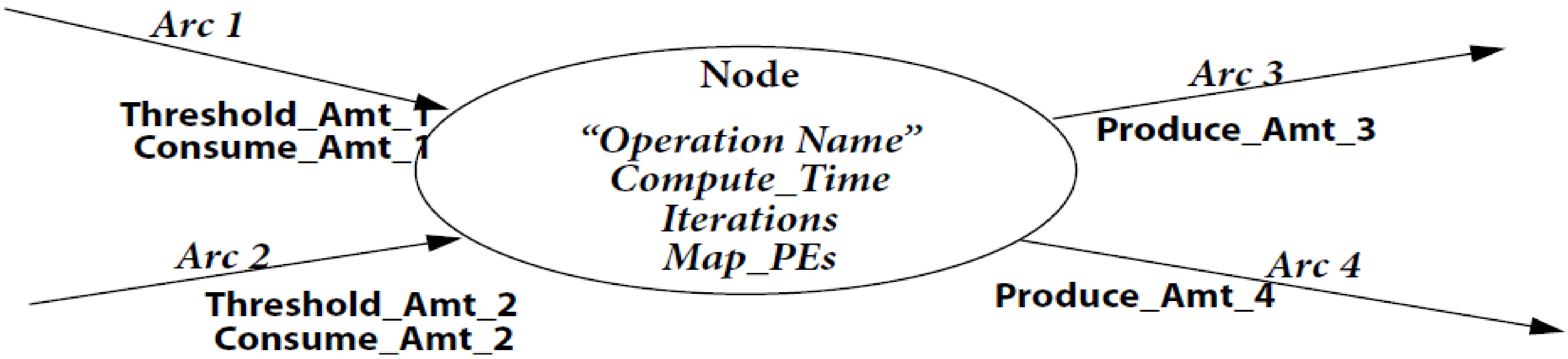


System Architecture Simulation



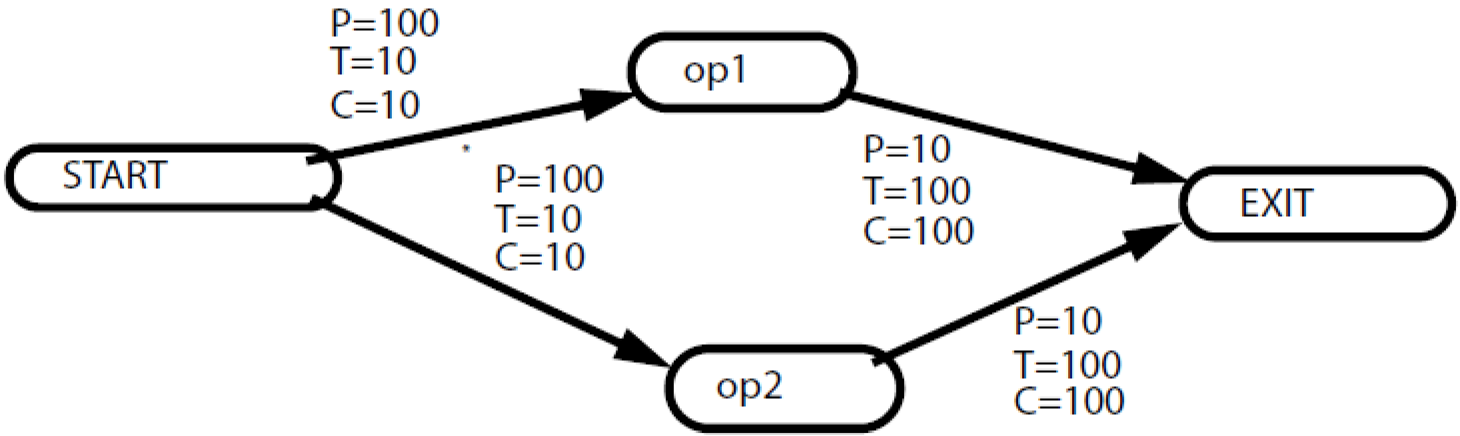
Application Data Flow Graph

- Flow graphs conveniently describe many applications.



- Node is said to be ready to *fire* when all input-arc queue-lengths $\geq$ arc *thresholds* and output-arc queue-lengths are < limits.
- To fire, a processor element must be available to execute the task.
- When node *fires*:
  - *Consume_Amount* data is removed from each input arc.
  - Node task begins *Iterations*-executions (normally, iteration=1).
- Each execution iteration lasts *Compute_Time uS*.
- After each execution iteration, *Produce_Amount* of data is placed on each output arc.
- Task node can be assigned or mapped to execute on specific processor elements.

Example Data Flow Graph

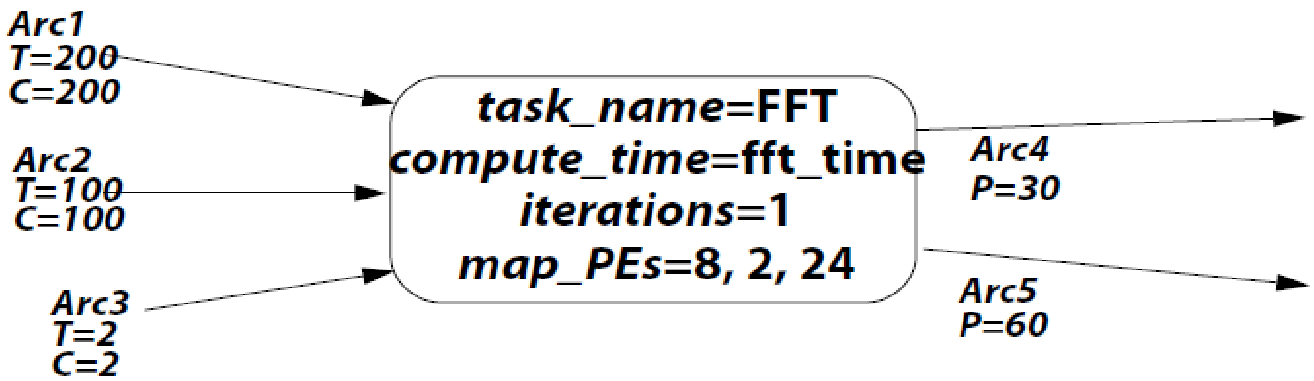


P=produce_amount, T=Threshold_amount, C=consume_amount.

*START produces 100 on arc each time it fires.
OP1 consumes 10 from arc each time it fires.

Software Application Program

- The application flow graph must be partitioned, mapped, and scheduled onto the available processor elements of a candidate architecture.
- The result is to be used to *drive* or *control* the PE hardware models.



- A typical flow graph node can be expressed as a group of simple instructions.

RECV(arc1, 200 bytes)
RECV(arc2, 100 bytes)
RECV(arc3, 2 bytes)
COMPUTE(fft_time, FFT)
SEND(arc4, PE4, 30 bytes)
SEND(arc5, PE14, 60 bytes)

- These pseudo-code instructions are interpreted by the PE hardware models.

Software Application Programs Model

Four Primary Instruction Types:

RECV(message_ID, message_length)
SEND(message_ID, destination_PE, message_length, priority)
COMPUTE(time_delay, task_name)
LOOP

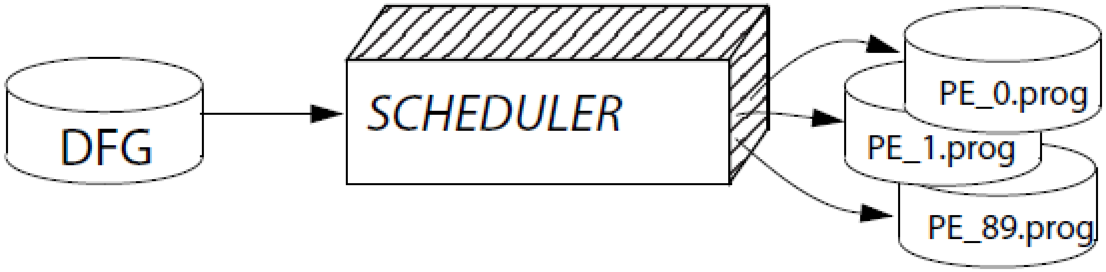
Example Program for /Board1/ce1:

recv 10 16384 -- Get input data for one range-pulse
compute 2160.0 Polarization1_Range1 -- Perform range-processing on data
send 1 2 2048 2 -- Distribute corner-turn data to neighboring send 1 3 2048 2 -- ...
. . .
send 1 8 2048 2
recv 10 16384 -- Get input data for new range-pulse
compute 2160.0 Polarization1_Range9 -- Perform range-processing on data
send 1 2 2048 2 -- Distribute corner-turn data to neighboring . . .
recv 8 131072 -- Get corner-turn data
compute 1730.0 Polarization1_Azimuth1 -- Perform azimuthal processing for one image raster

CSIM's SCHEDULER tool takes care of
the complex message indexing

Software Application Programs Model

- It would be very tedious to manually convert flow graph descriptions into pseudo-code programs for each PE.
- Making the message-IDs match in corresponding PE files can be very Complex. (*message indexing*)
- An automatic pseudo-code generator is available to ease this task.
- It is called the **SCHEDULER**.
- The SCHEDULER converts DFG descriptions into pseudo-code programs.



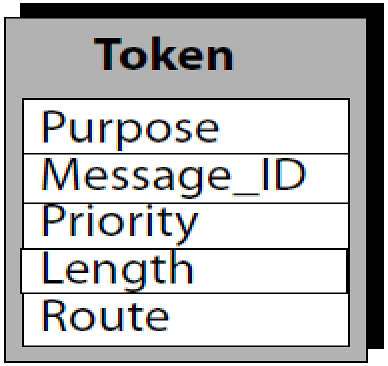
- Map_PE assignment of each node is optional.
- If you do not assign it, the SCHEDULER will assign automatically.
- The Partitioning/Mapping/Scheduling problem is NP-complete.
- These tools are aids to finding optimal solutions.

Computer Hardware Architecture Models

- Hardware architectures are composed of building-block models.
- Generic models of:
  - Processor Elements
  - Memories
  - Buses
  - Crossbars
  - I/O units
- Many specific variants of these have been generated for special purposes.
- All are based on common message token definition for interoperability.

Message Token Definition

Goal: Represent Message Transfers Through Circuit-Switched Protocol Network By Using Tokens.
Must account for message transfer time as function of traffic.
Note: Circuit-switched protocol exhibits blocking: it affects performance.




```
enum message_type { REQ, ACK, NACK, PREEMPT };
struct message_struct
    {
        int src, dst, mid;
        enum message_type purpose;
        int length;            /* message length in bytes */
        int packet_length;
        int route_list[MAX_ROUTE], route_index;
        int priority;
        float launch_time;
        int mbody;
    };
```

<u>Approach 1: Six Token Types</u>(purpose)

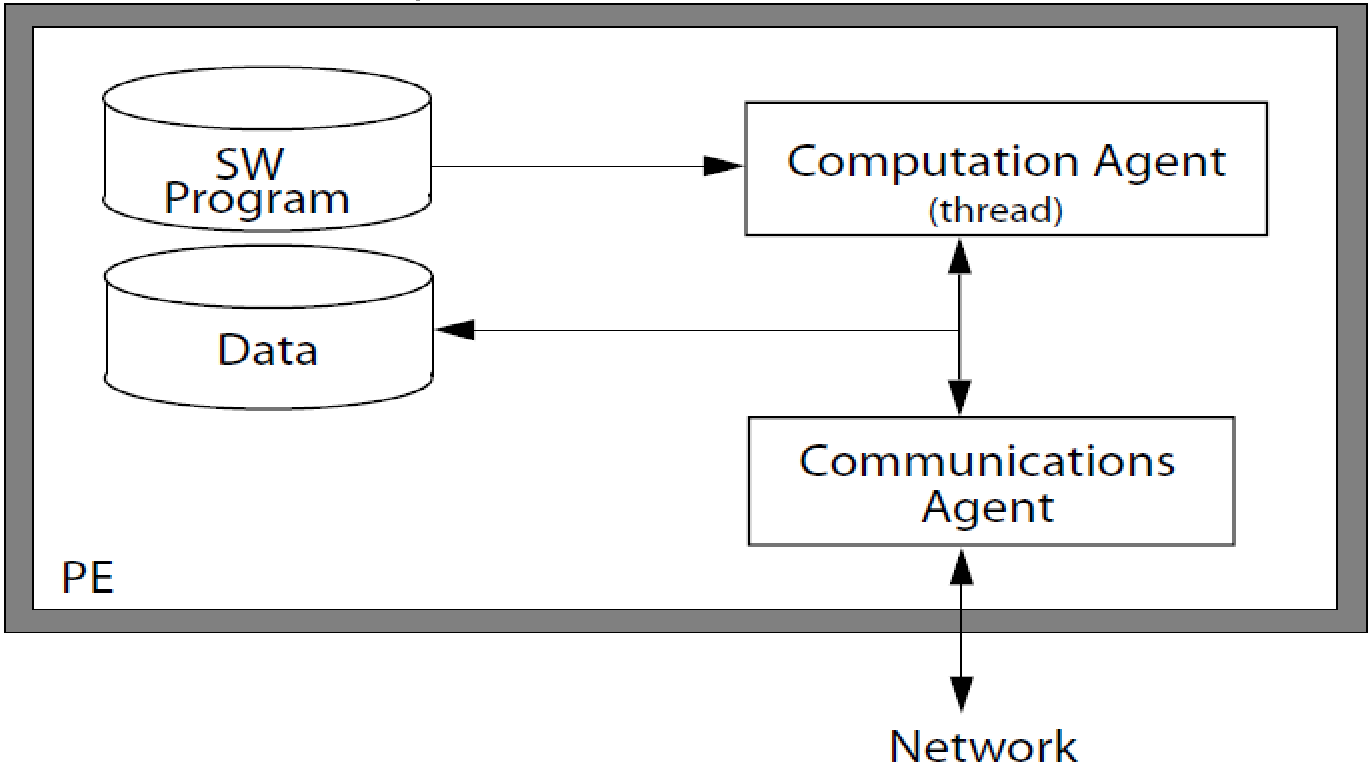
1. REQ - Request path through network
2. ACK - Grant request, path allocated
3. NACK - Request blocked
4. DATA - Data begins moving
5. DONE - Transfer complete, reallocate path
6. PREEMPT - Transfer preempted

<u>Approach 2: Four Token Types</u>

1. REQ - Request path through net
2. NACK - Request blocked
3. DONE - Transfer done, reallocate path
4. PREEMPT - Transfer preempted

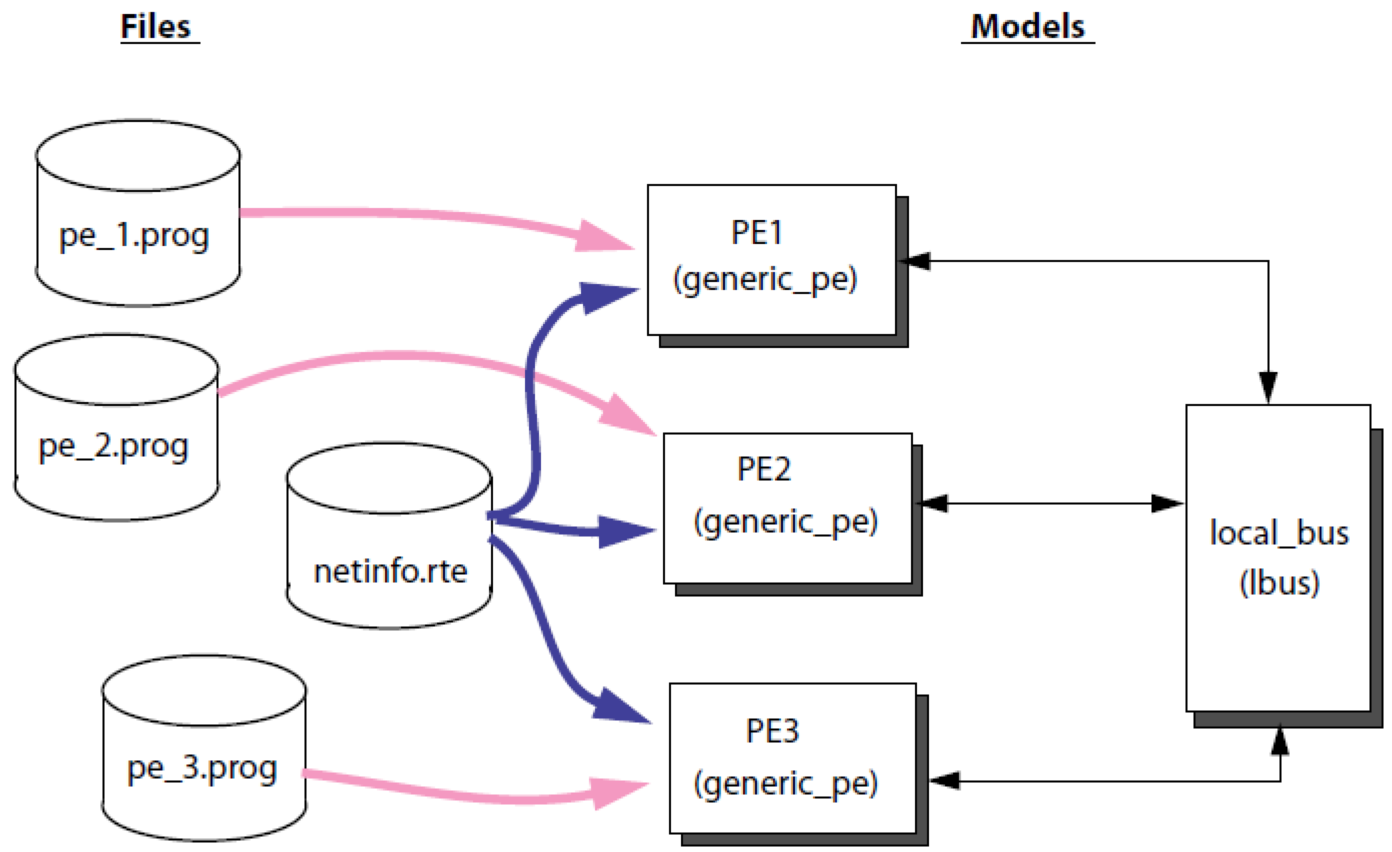
Processing Element Model

Simple abstract PE model



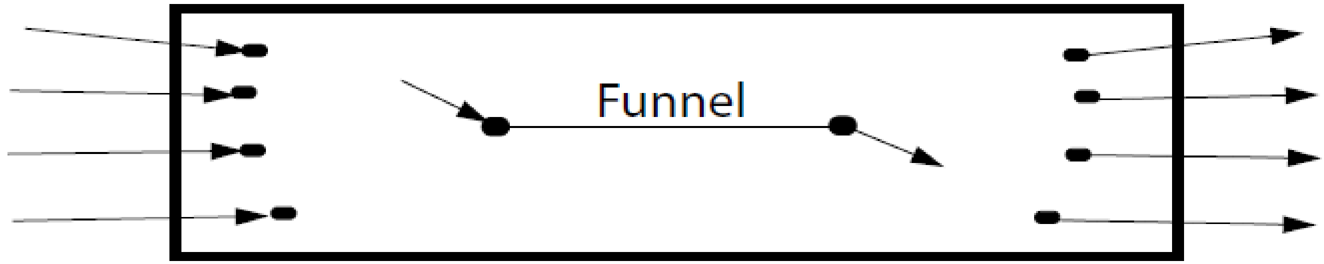
- For Architecture Performance Modeling
- Local memory for storage of local data and software.
- Two concurrent processes:
  - Computation agent interprets application software.
  - Communication agent provides reliable asynchronous reception and transmission of messages through network.

System Architecture Model



- Processor Element models read respective files at start-up:
  - Program files,
  - Routing file.
- Bus Elements do not read any files.

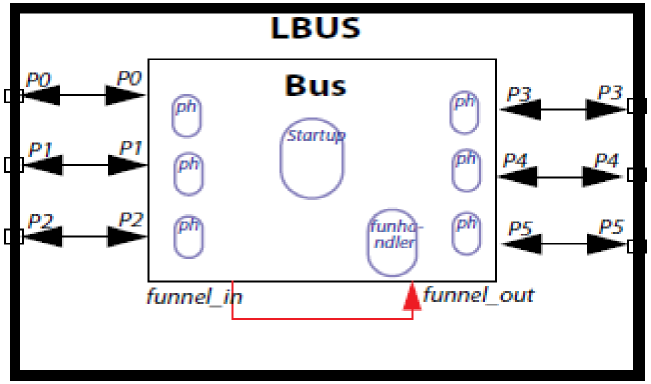
Generic Bus Model



Actual CSIM code for a Bus Model

```
/*****************************************************************/
/* CSIM Model of Generic Local Bus (LBUS)                        */
/*                                                               */
/*  This models a common bus by funneling all incoming packets   */
/*  one-at-a-time through a single internal queue.  Each input   */
/*  port of the bus has a process waiting for a packet to        */
/*  arrive.  When a packet arrives on the port, the process      */
/*  pushes the packet onto the "funnel" queue.  Concurrently     */
/*  arriving packets pass through the funnel one-at-a-time and   */
/*  are routed out to their respective destination port.         */
/*                                                               */
/*  One process handles the funnel-queue by receiving messages   */
/*  on the internal queue and dispatching them to the            */
/*  appropriate output ports.                                    */
/*****************************************************************/

DEFINE_DEVICE_TYPE:     Bus   /* N-port data funnel */

 PORT_LIST( p0, p1, p2, p3, p4, p5 );

  /* Local variables to device instance. */
  double link_usage;
  char **port_names;
  int nports;
  float times_in, times_out;
```

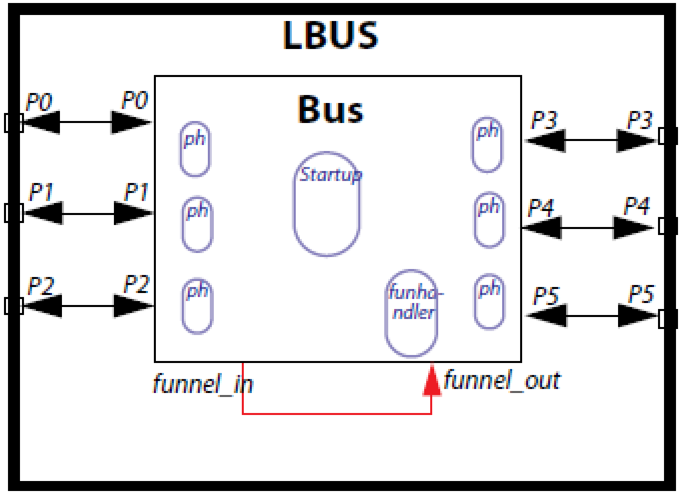
Generic Bus Model (continued)




```
DEFINE_THREAD:  start_up
  {  /* Initialize statistics reporting */
    times_in = 0.0;
    times_out = 0.0;

    port_names = list_in_ports( &nports );

    /* Start the internal funnel-queue handler. */
    TRIGGER_THREAD( port_handler, 0.0, 0 );

    /* Start the internal funnel-queue handler. */
    TRIGGER_THREAD( f_handler, 0.0, 0 );

    WAIT( &CSIM_EndOfSim, NONQUEUABLE );

    csim_printf("Utilization = %f percent\n", (times_out - times_in) / CSIM_TIME);

  }
END_DEFINE_THREAD.


DEFINE_THREAD:  port_handler        /* Port handler process.  */
 {
  struct header_struct *message;   /* Local variables to thread instance. */
  int length;

  while (1)
  {
    RECEIVE_ANY( port_names, &message, &length );
    times_in = times_in + CSIM_TIME;
    SEND( "funnel_in", message, message->packet_length );
  }
 }
END_DEFINE_THREAD.
```

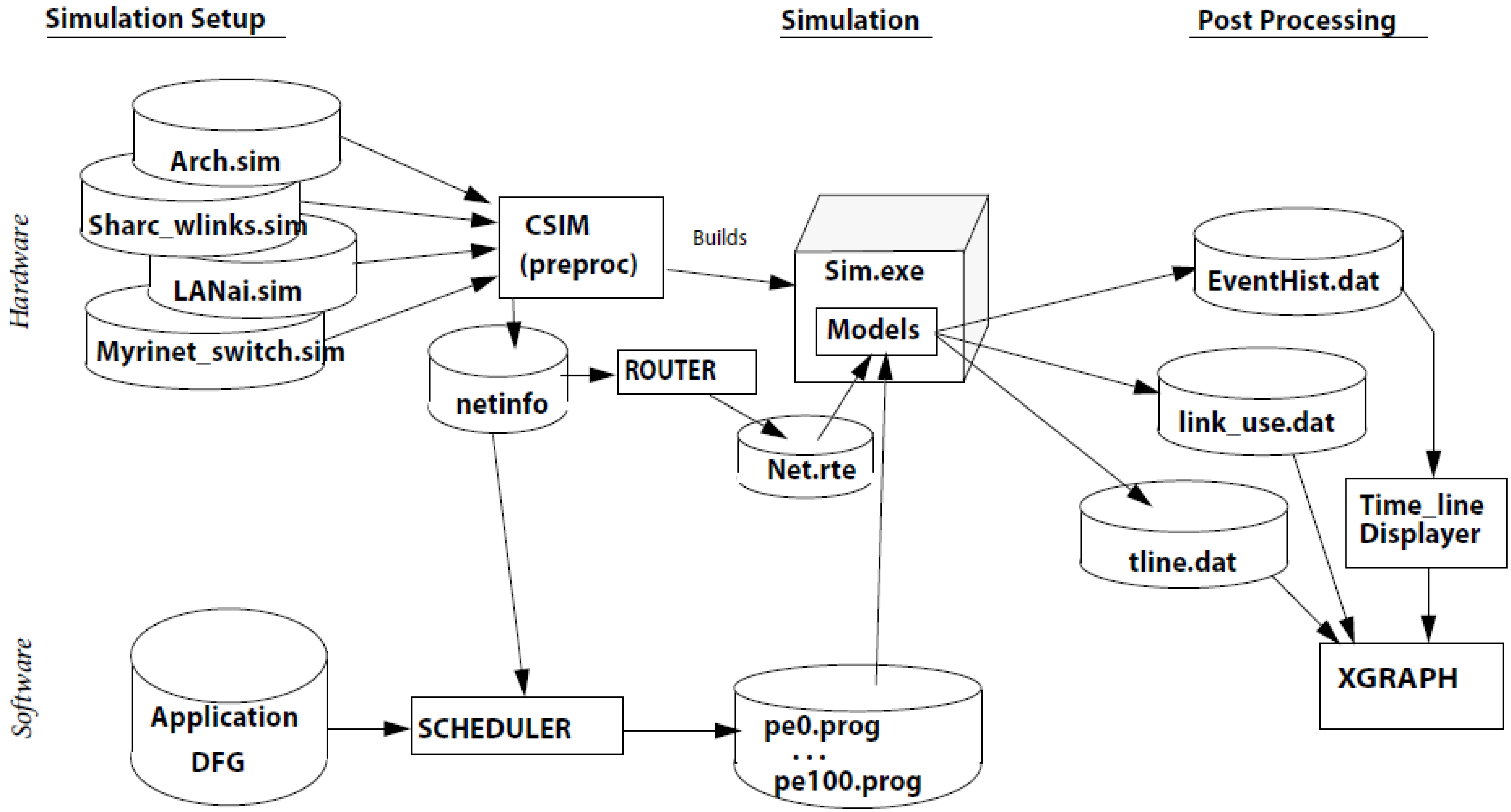
Generic Bus Model (continued)

```
DEFINE_THREAD:  f_handler
 {
  struct header_struct *fmessage;      /* Local variables. */
  int flength, pnum;

  while (1)   /* Do forever. */
   {
    RECEIVE( "funnel_out", &fmessage, &flength );    /* Wait for full packet. */
    times_out = times_out + CSIM_TIME;               /* Record usage stats. */
    pnum = fmessage->route_list[ fmessage->route_index ];  /* Handle routing. */
    fmessage->route_index = fmessage->route_index + 1;     /* Increment packet's hop. */
    SEND( port_names[pnum], fmessage, flength );    /*Send packet out respective port*/
   }
 }
END_DEFINE_THREAD.

END_DEFINE_DEVICE_TYPE.
```



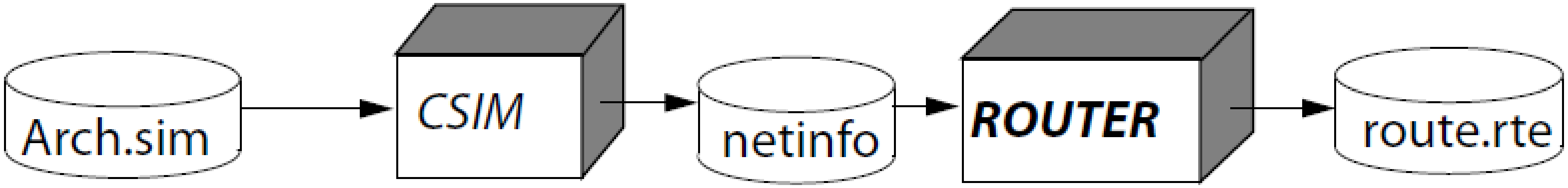
Tool Flow

Performance Modeling Simulation Environment



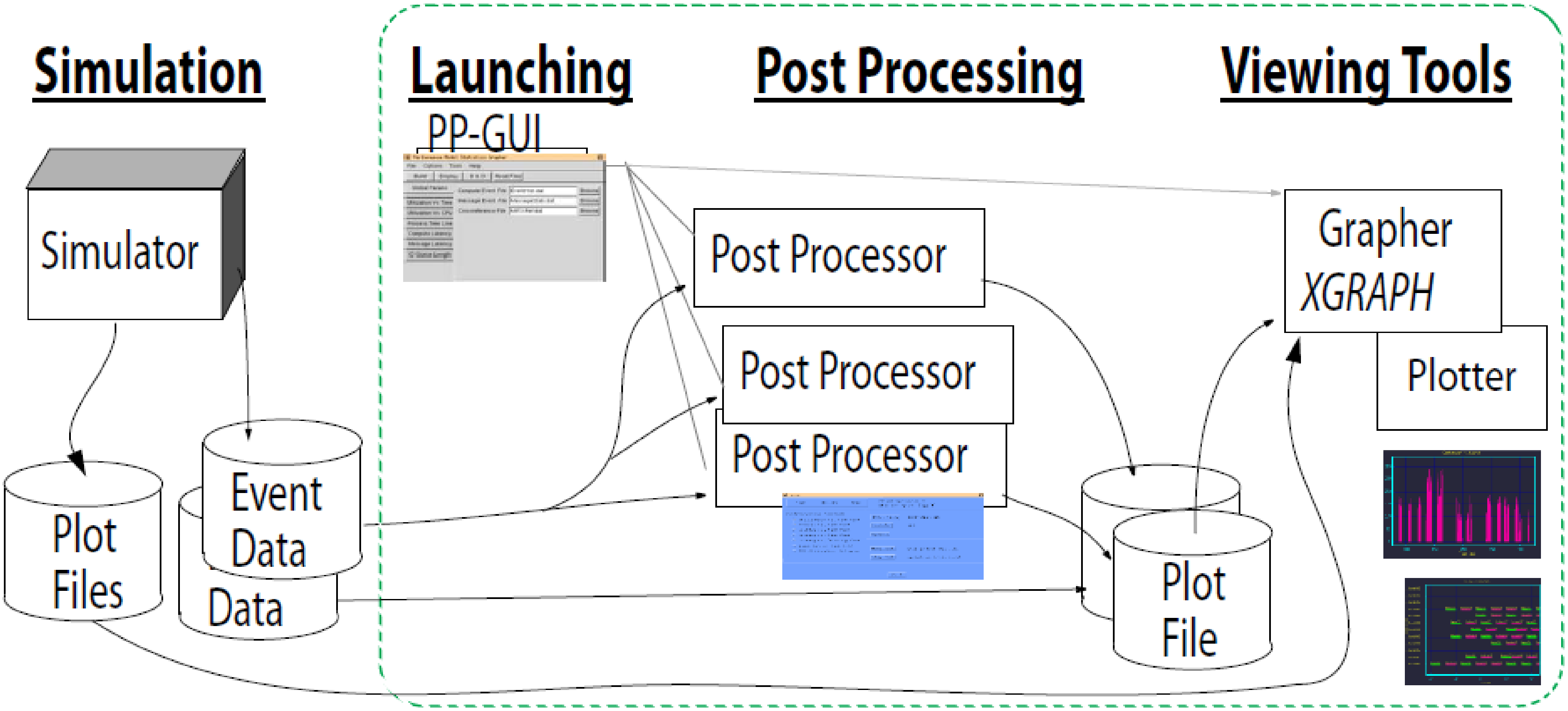
Routing Tables

- Generating a routing path to reach another PE would be time-consuming if conducted during runtime, because common paths are used repeatedly and architectures are complex.
- The Perf.Mod.Lib models use pre-prepared routing tables.
- The number of entries in a routing table is: NxNxM
  Where N is the number of devices in the system,
  and M is the number of alternate paths to store.
- It would be tedious to correctly prepare such tables manually.
- An automatic router is available, called: ***ROUTER***



- ROUTER interprets your architecture description via a CSIM netinfo file.
- Produces routing table in the format expected by Perf.Mod.Lib. PEs.

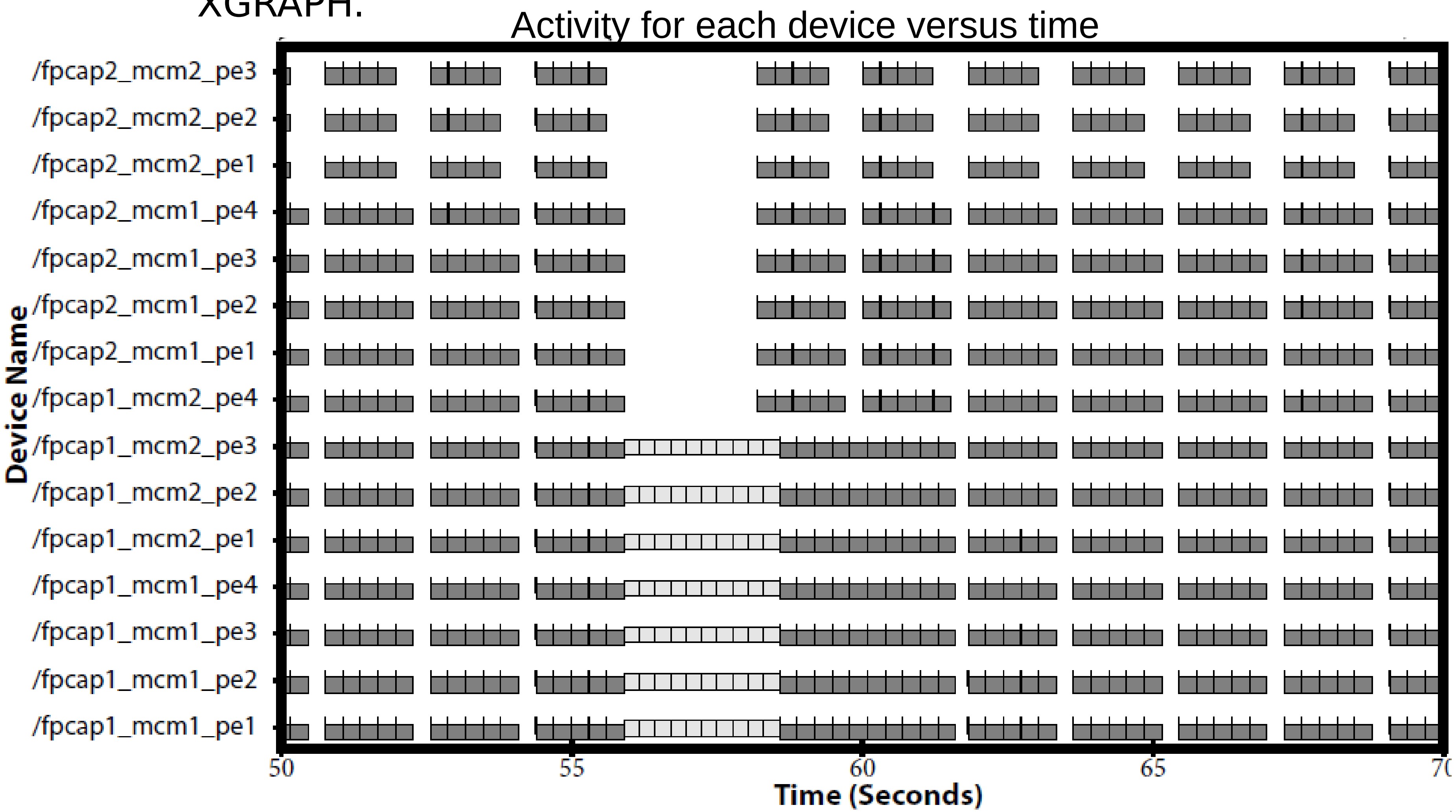
Post Processing Results



– Several methods to view results, several pathways.
– You can customize new post-processors.
– Results can be archived, analyzed later, and compared.

Viewing Simulation Results

– Example time-line from XGRAPH.



Development Techniques

– Distributing Simulations
– Animating
– Annotating
– Debugging
– Log-files
– Handling threaded software
– Modifying routing tables

Model Calibration - Validation

Process for accurate abstraction:

Estimates → **Model(s)**
Parametric Expressions,
coef, curve fitting

Calibrate

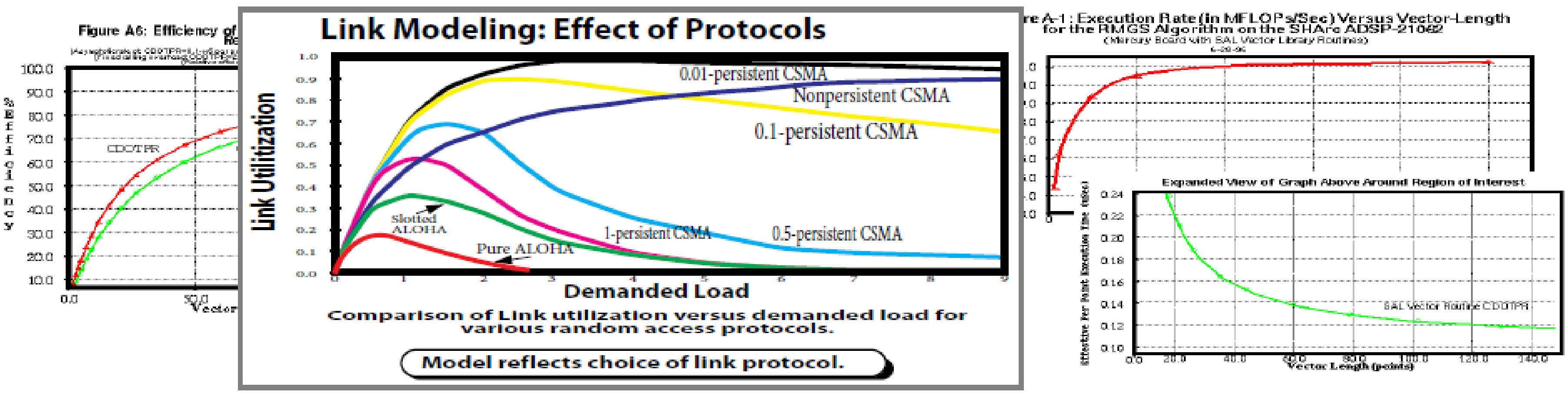
-

Benchmark(s) or Detailed Model(s)

Application Benchmarks
Generalized Canonical Benchmarks
Applications
Std. benchmarks
Published tables

Communication Benchmarks
A, B, C, D
Published tables

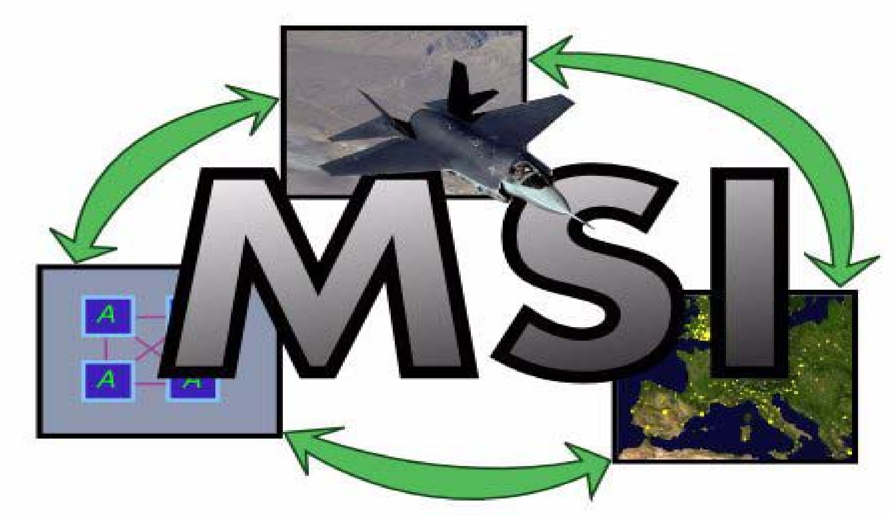
Assess linear additive/predictability
vs. chaotic elements



Accuracy + Abstraction

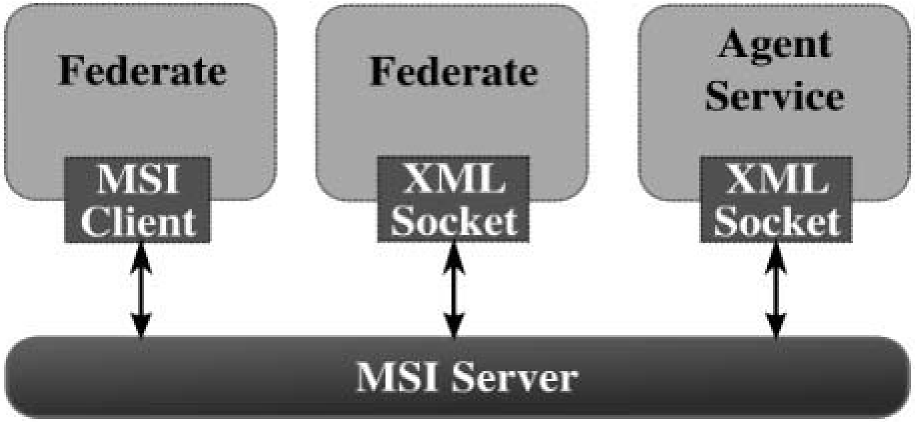
Multi-Simulation Interface

(MSI)



- A simulation interconnection engine.
- Like MSCO High Level Architecture HLA.
- Open Freeware - Gnu Public License.
- See: *http://msi.sourceforge.net/*

MSI Information



- Based on XML socket stream.
- Usable from any programming language.
- C, C++, Java, Ada, Perl, etc.
- No library dependencies.
- Cross-platform code, portable to all major OS platforms (Linux-PC, Solaris, Irix, HP-UX, Mac OS X, MS Windows, FreeBSD, etc.).
- Provides managed federation start-up (join) control.
- MSI is a single executable file.
- Distributed with example code for simulator (federate) side interface.

MSI Description Concepts

CSIM

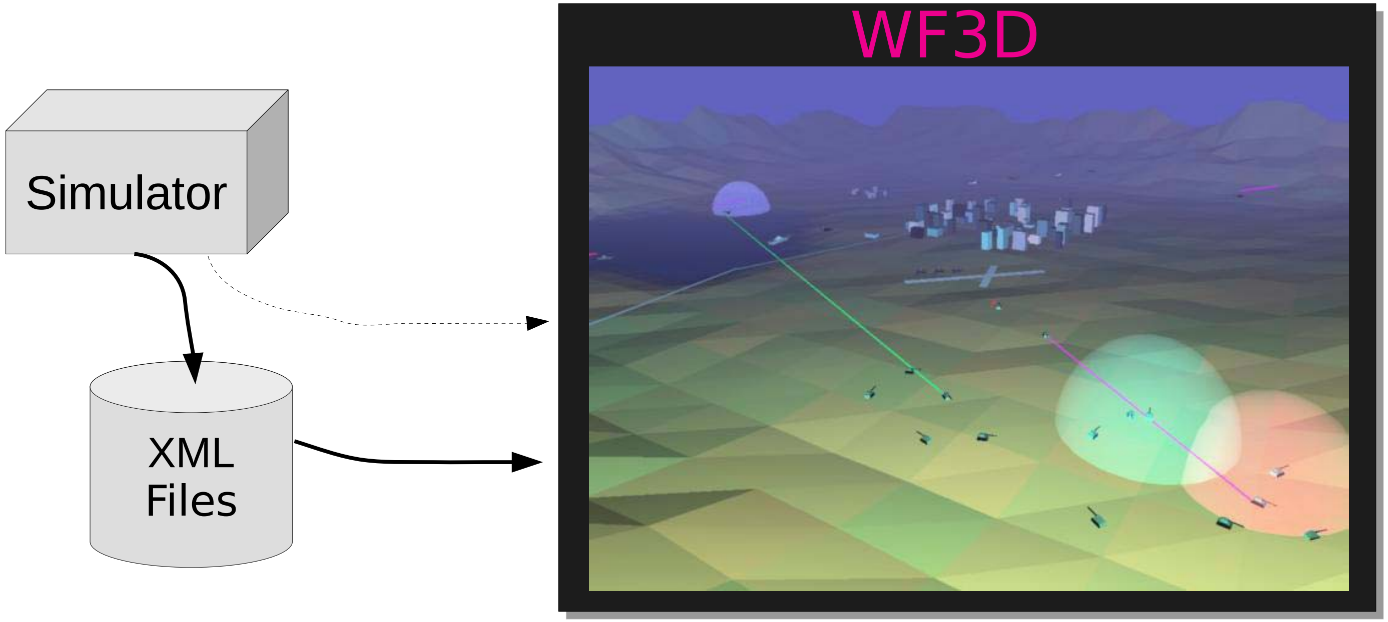
Direct face-to-face mappings

- Data exchange formats
  - Instead of forcing clients to convert all shared data to neutral formats, Meta-data is added to each exchange, enabling envelope parsing; ==> Verbose.
  - Avoids high processing and bandwidth costs for transforming and packing all data.

- MSI Rosetta Stone approach
  - Instead of transforming data on input, MSI leaves data in original form for efficient transfer.
  - MSI client-side library transforms data to best local form.
  - If receiving simulation can parse incoming data, then no translation overhead incurred.
  - XML enables context insensitive parsers and transformability.
  - XSLT can specify arbitrary transforms; ==> Supports direct inter-ontology mappings .

WinFrame 3D - Viewer (WF3D)

- For visualizing simulated platform locations, movements, and activities.



- Accepts input from files or sockets.
- XML format.
- See documentation, tutorial, and examples on CSIM web-page under WF3D

WinFrame 3D - Viewer

1. Specify Initial Camera Settings

– View position, field of view, depth of field.

```
<set_camera>
    <frustum fov="", near_field="", far_field="" />
    <position x="", y="", z="" />
    <pointat x="", y="", z="" />
</set_camera>
```

2. Define the object types

– Type *humv* shape and color is …

```
<def_obj> type_name
    <col r="" g="" b="" t=""/>
    <quad> <vrt x="" y="" z="" /> .... </quad>
    <image> image_filename
      <vrt x="" y="" z="" /> <vrt x="" y="" z="" />
      <vrt x="" y="" z="" /> <vrt x="" y="" z="" />
    </image>
    …
</def_obj>
```

3. Instantiate Objects -

– Object Q is of type humv at location x1,y1,z1.
– Object W is of type humv at location x2,y2,z2.

<inst_obj name="" kind="" x="" y="" z="" xang="" yang="" zang="" />

4. Move Objects -

– Move object Q to position x,y,z by time T.

<mo obj="" T1="" T2="" x="", y="", z="" />

Vehicle Platforms Terrains (VPT)

Spatial Modeling Library - VPT

- Manages platform positions over time.

- Way-point based.

- Services Provided:
  - VPT_Get_Neighbors(myplatform, radius);
    Returns list of nearby neighboring platforms.
  - VPT_Get_Position(platform, time, &x, &y, &z);
  - VPT_Get_Heading(platform, time, &heading);
  - VPT_Get_Velocity(platform, time, &vx, &vy, &vz);
  - VPT_Set_Velocity(platform, time, vx, vy, vz);
  - Coordinate conversion routines.
    UTC, Cartesian (x,y,z), Lat-long, polar, ...

- Compatible with ScenGen - Scenario Entry Tool (SET),
  and WF3D - Scenario Visualization Tool.

Debugging Tips

- Raise verbosity. Most CSIM tools have a *-v* command-line option.

    csim -v 1000 testarch.sim

- Use Debugger. View variable interactively. Stops simulation on error line.

    gdb sim.exe
    run
    print xyz

  Or, use *ddd* for graphical debugger.

- Use -trace. CSIM inserts unique Debug xx printf between every line of models.

    csim -trace arch.sim

- Capture output to log file. Search with grep/more/editor.

    sim.exe > log

- Use -nomarks. Errors reported relative to intermediate file out.c.

    csim -nomarks arch.sim

- Raise model and/or simulator verbosities.

General Utilities

General Utilities Directory:

- A set of utilities especially useful for working with simulation data files and complex projects.
- Under tools for each platform.

  Ex. *$CSIM_ROOT/tools/linux_2.3/general_utilities*

Tools List:

- **c2html** - Convert C-code to HTML, routine, index, hyper-text, call-tree, ..
- **draw_tree** - Draws a printable directory tree, showing space used, dates, ..
- **disk_usage** - Shows space used by directory branches.
- **nesting_list** - Shows nesting level of code. Produces annotated listings.
- **scengen** - Scenario Generator, entry tool, for VPT and wireless models.
- **line_dif**f - Compare files and show line by line differences.
- **compare_dir** - Compare whole directories to other directories.
- **filter** - Globally replace phrases without editing files.
- **scz_compress** - Simple lossless compression. CSIM tools decompress on-the- fly.
- **scz_decompress** - Opposite of above.
- **xyz2wf** - Plot matrix data in 3D by converting to WF3D format.
- **SimDif**f - Compares diagram versions.

Modeling Workshop

– Enhancing hardware-architecture models,
– Debugging models and building simulation,
– Extending application-scenario graphs
– Running simulations, and analyzing results

Documentation

- Current documentation, examples, new, & information are maintained on-line at:

www.csim.com

CSIM Web Page