

Multi-Level and Multi-Formalism Modeling of Ultra-Large Networks

William H. Sanders

Performability Engineering Research Group
Coordinated Science Laboratory and Electrical and
Computer Engineering Department
University of Illinois at Urbana-Champaign
whs@crhc.uiuc.edu



<http://www.crhc.uiuc.edu/PERFORM>

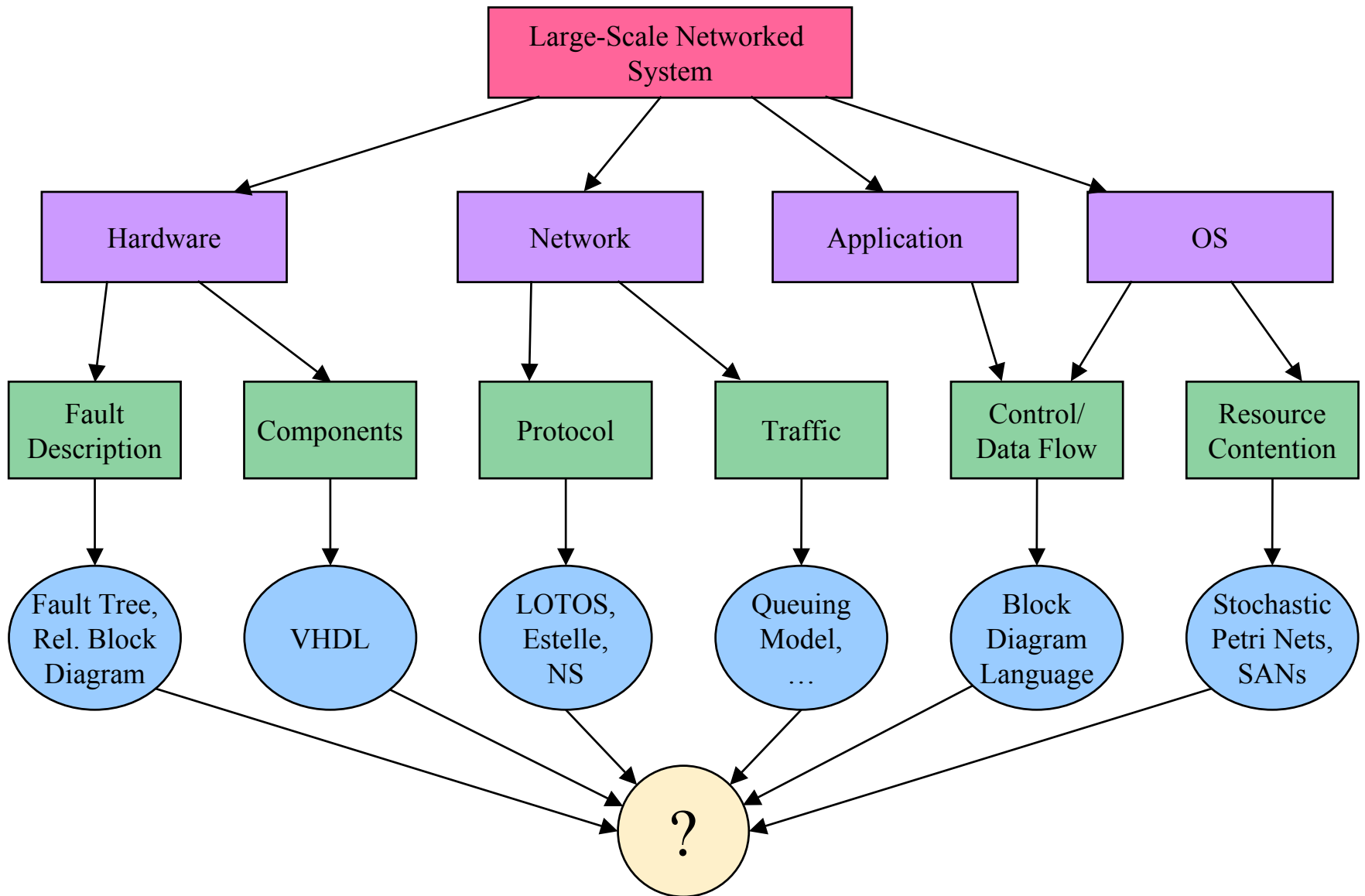
NSF Workshop on Modeling and Simulation of Ultra-Large Networks:
Challenges and New Research Directions, November 19-20, 2001

Project funded in part by the Motorola Center for High-Availability System Validation, under the umbrella of the Motorola Communications Center, and National Science Foundation Next Generation Software Program

Integrated Modeling Frameworks are Needed!

- No single measure (or class of measures) is sufficient to capture the quality of service experienced by a user of a large-scale networked system
 - Performance, Dependability, Survivability, Intrusion-Tolerance, and Security are all valid concerns, and need to be integrated to into high-level, user oriented measures.
- No single formalism is best for representing all parts of a large-scale networked system
 - Computer hardware, networks, protocols, and applications each call for a different representation
 - Even within a “class” of application, different industry segments use very different ways of representing a particular design
- No single solution method is adequate to solve all models
 - Discrete-event simulation is efficient in many cases, but is extremely slow in others (e.g., significant, but rare events (like faults and buffer overflows), or high system complexity)
- Research in new modeling methods and tools is significantly hampered by the close link between model specification and model solution methods, and the closed nature of existing tools

Modelers Need Heterogeneous Models



State of the Art: Single Formalism Tools

- Many performance/dependability evaluation tools have been developed that provide a single modeling formalism, and support multiple solution methods, e.g.,
 - **Queueing networks**, e.g., DyQN-Tool, HIT, LQNS, QNAP2, RESQ, and RESQME. Most tools support both simulation and product-form based solutions.
 - **Stochastic Petri nets and extensions**, e.g., DSPNExpress, ESP, GreatSPN, HiQPN-Tool, QPN-tool, SPNP, SPN2MGM, SPNL, SURF-2, TimeNET, and *UltraSAN*. All tools support analytical/numerical solution; some support simulation.
 - **Stochastic Process Algebras**, e.g., EMPA, Dragon, PEPA Workbench, TIPPTool, Two Towers, and Spades. All tools support analytical/numerical evaluation; some support simulation.
 - **Other modeling approaches**, sometimes tailored to a specific application domain, e.g., DEPEND, DEVS, Figaro, HARP, HIMAP, NS, Peps, SAVE, SPE*ED, and TANGRAM-II.
- ⇒ In most cases, each tool has multiple model solution methods (recognizing the fact that no single solution method is sufficient in all cases), but a single model specification method.

State of the Art: Combination of Multiple Tools in a Single Software Environment

- Several tools have been constructed that facilitate the combination of multiple tools into a single environment, e.g.,
 - **IMSE** (Integrated Modeling Support Environment) [Pooley 91]
 - Contains tools for modeling, workload analysis, and system specification
 - **IDEAS** (Integrated Design Environment for Assessment of Computer Systems and Communication Networks) [Fricks 96]
 - Provides user interface to multiple tools without requiring a user to learn multiple interface languages and output formats
 - **Freud** [van Moorsel 98]
 - Aims similar to those of ISME and IDEAS, but focuses on providing a uniform interface to a variety of web-enabled tools
- ⇒ Focus is on building a common graphical interface for accessing multiple tools and a common method for reporting results.

State of the Art: Integrated Modeling Frameworks

- Integrated modeling frameworks aim to define an environment that can accommodate multiple modeling formalisms, one or more ways to combine models expressed in possibly different formalisms, and multiple model solution methods, e.g.,
 - **SHARPE** [Sahner 86, Sahner 96]
 - Models expressed as combinatorial models, directed acyclic graphs, Markov and semi-Markov models, product-form queueing networks, and GSPNs can be solved, and can exchange results expressed as exponential-polynomial distribution functions
 - **SMART** [Ciardo 96, Ciardo 97]
 - Models expressed as SPNs, “software modeling language,” and Markov chains can be solved, and can exchange results, possibly repeatedly, using fixed-point iteration
 - **APNN toolbox** [Bause 98]
 - Models converted to common “abstract PN notation” (APNN) and can be solved in a variety of quantitative and functional analysis methods.

Integrated Modeling Framework Requirements

- Provide a method by which multiple types of measures (performance, dependability, survivability, intrusion-tolerance, and security) can be combined into a high-level framework.
- Provide a method by which multiple, heterogeneous models can be composed or connected together, each representing a different software or hardware module, component, or view of a system
- Permit models to interact with one another by sharing state, events, or results and be scalable, in the sense that the solution of an entire model should be possible at a cost lower than for an equivalent unstructured model
- Support multiple modeling formalisms, as well as methods to combine models at different levels of resolution
- Support multiple model solution methods, including both simulation and analysis
- Be extensible, in the sense that it should be possible to add new modeling formalisms, composition and connection methods, and model solution techniques to a tool that implements the framework without changing existing tool components

Möbius Project Research Goal

- Development of tools to predict the performance, dependability, and performability of distributed computing/communication systems
 - Such systems are complex combinations of:
 - Computing hardware
 - Networks
 - Operating systems
 - Workload/Fault-Load/Attacker Profile-Load (X-load)
 - Software

(First goal was *not* to prove logical system properties, although work is now ongoing to do this in cooperation with the University of Twente.)

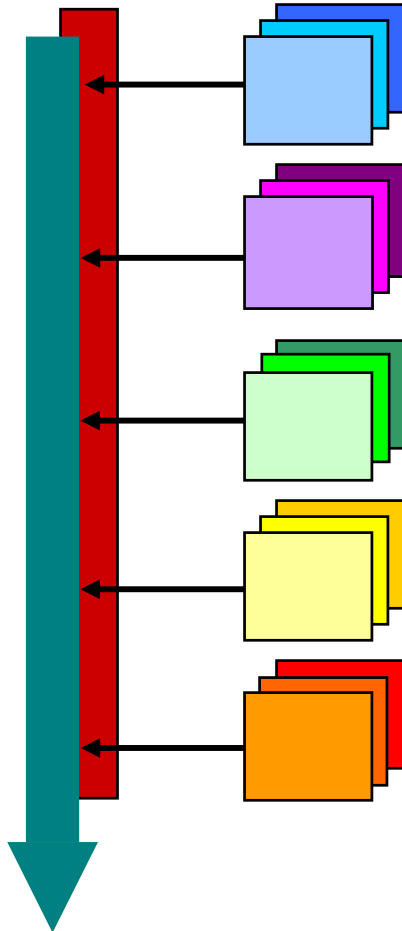
- We believe such tools can be realized by:
 - Developing a framework/tool that supports multiple modeling formalisms, at multiple levels of detail and abstraction, and multiple model solution methods
 - Developing new model representation and solution methods (within the framework, and implemented in the tool) that scale well with increasing system complexity

The Möbius Framework ...

- Can express most existing modeling languages
- Retains the ability for efficient solution
- Facilitates heterogeneous modeling
- Is a vehicle for researching new model composition, connection, reward specification, and solution methods

Model Categories in the Möbius Framework

Submodel Interaction



Framework Component

Atomic Model

Composed Model

Solvable Model

Connected Model

**Study Specifier
(generates multiple
models)**

Example Formalisms

DSPN, GSPN, Markov chain,
Queueing Network, SAN, SPA,
other SPN extensions,
Domain-specific formalism

Graph interconnection
Kronecker Composition (SAN),
Replicate/Join, SPA
Domain-specific formalism

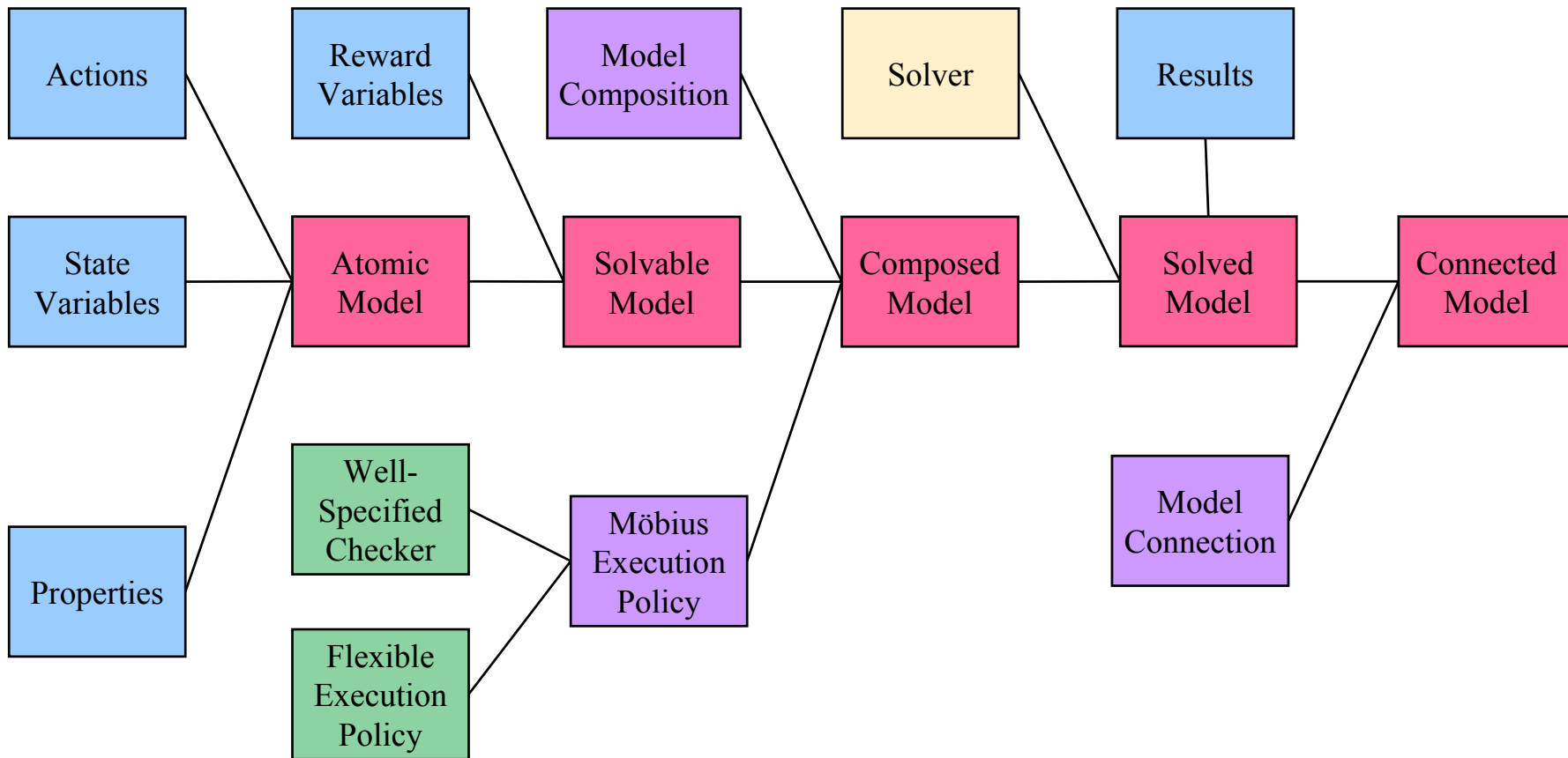
Rate/Impulse reward variables
Path-based reward variables
Domain-specific formalism

Fixed-point governor
Acyclic model composer

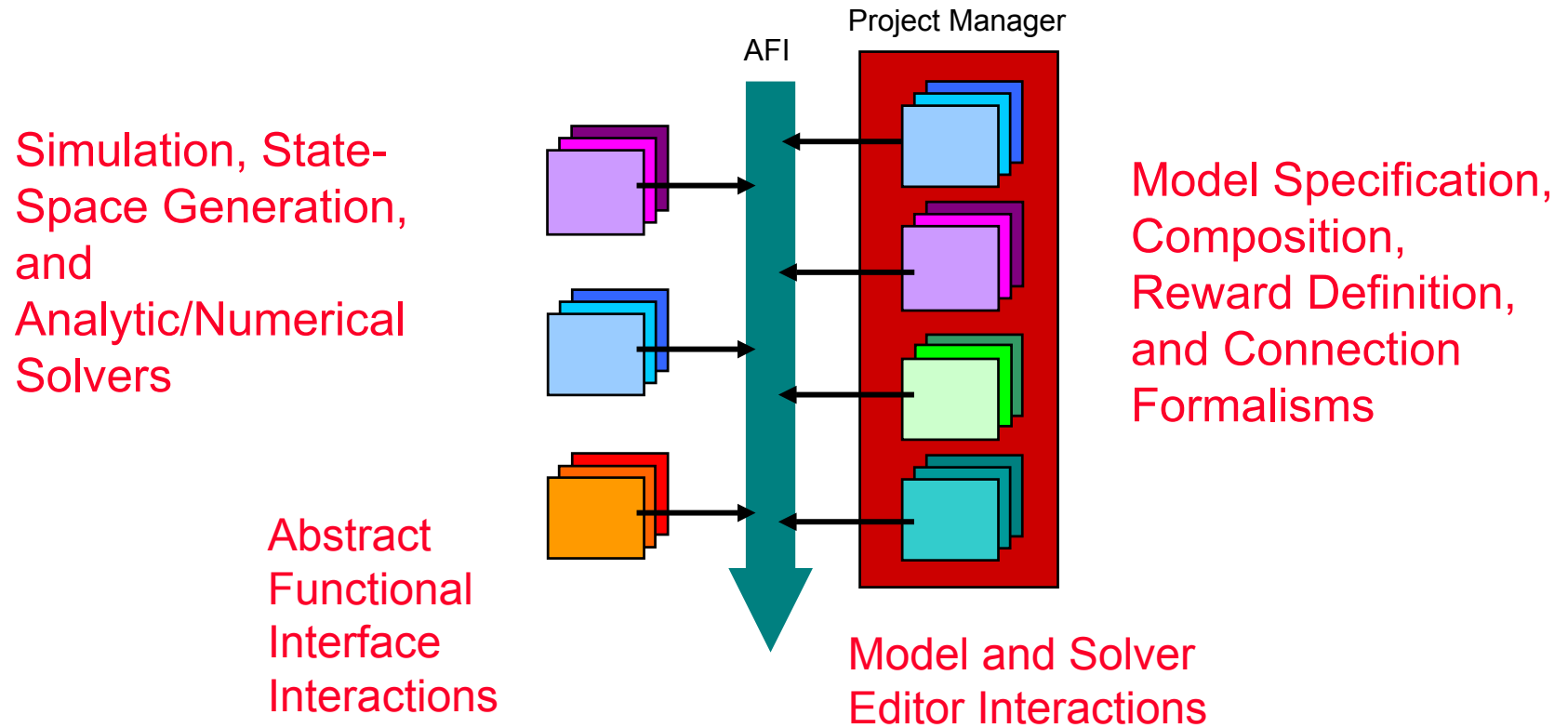
Range and Set Variation
Non-linear optimizer

Model Specification

Framework Components Roadmap



Abstract Functional Interface Facilitates Interaction of Models and Solution Engines



- The **abstract functional interface** allows models to affect each other and be acted on by solvers without understanding model semantics
- A **project manager** maintains consistency when constructing new models, performance/dependability variables, and studies from existing models

Technical Details: Abstract Functional Interface (AFI)

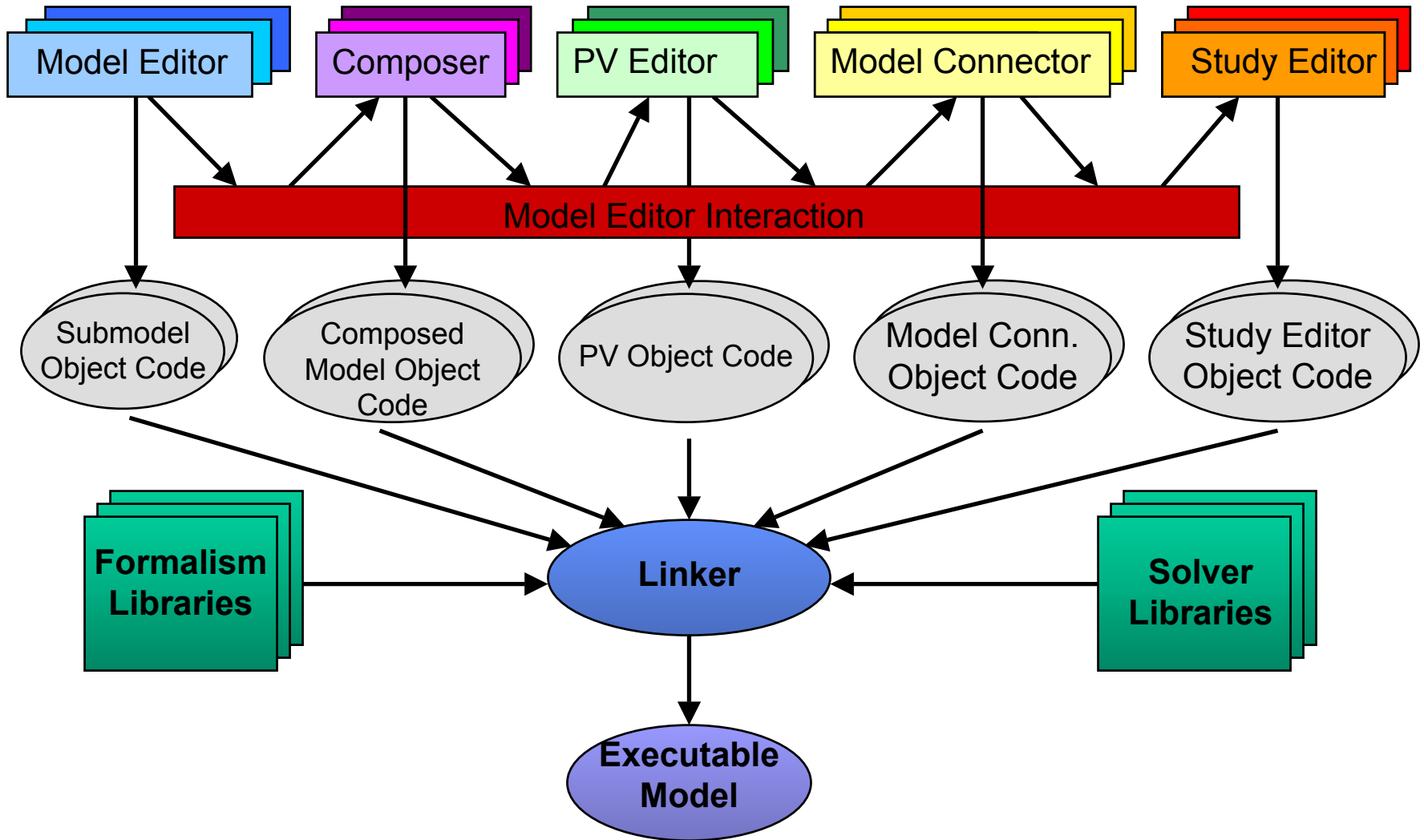
The **abstract functional interface** is a set of methods defined on a set of “base classes” that all models must implement to interact within the Möbius framework.

- The AFI acts as a **communication interface between multiple, possibly heterogeneous, models**.
- **Solvers communicate with models by calling methods in the AFI**.
- **The AFI can preserve special properties** of models implemented in particular formalisms since it only specifies an interface, not internal representation, unlike the integrated framework approach which converts models to a universal formalism.
- **The AFI makes Möbius extensible**, by hiding formalism-specific details from interacting models or solvers, making it possible to add modeling formalisms that interact with models described in other formalisms and solvers, without changing them.

Model Support of the Abstract Functional Interface: State Variables, Actions, and Properties

- Formally, a **model** in the Möbius framework is a set of “state variables,” a set of “actions,” and set of “properties”
- **State variables** “contain” information about the state of the system being modeled
 - They have a **type**, which defines their “structure”
 - They have a **value**, which defines the “state” of the variable
- **Actions** prescribe how the value of state variables may change as a function of time
- **Properties** specify characteristics that may effect the solution of a model
- Other models and solvers may request information regarding or change to state of a model’s state variables, actions, and groups via the abstract functional interface
- The format of this information is determined by the structure of a model’s state variables and attributes of its actions

Möbius Tool Architecture



Atomic Model Construction

- Each atomic model editor permits the specification of a particular atomic model formalism -- editors can be graphical or textual.
- Möbius toolkit provides Java building blocks for editor construction, easing editor implementation.
- Each editor must generate model representation that can be “executed” by Möbius solvers; note that the representation can be formalism specific.
- Together, the code emitted by the editor and formalism “library” must implement the AFI to models written in the formalism.

The image displays the UltraSAN/Möbius SAN Editor 1.1.1 interface, showing the construction of an atomic model for a processor and a test.

faulty_proc2: processor

The main window shows a Petri net diagram with places: queue, Ac_accessible, num_tasks, Ac_processing, and rea. Transitions are labeled: lg_available, Og_correct, and rea. The diagram shows the flow of tasks through the processor, from the queue to Ac_accessible, then to num_tasks, then to Ac_processing, and finally to rea.

AC_processing Case Distribution

The Case Distribution dialog shows the following code for Case 1:

```
if (num_tasks->Mark0 == 1)
  return(1.0);
else return(ok_prob);
```

test: marca

The main window shows a Petri net diagram with places: state01, state02, state03, and state04. Transitions are labeled: st01, st02, st03, and st04. The diagram shows the flow of the test through the states.

Timed transition attributes

The Timed transition attributes dialog shows the following settings:

- Name: TimedTrans11
- Number of transferring balls: 3
- Time distribution function: Exponential

The main window shows the following code for the test:

```
ar := 4.5;
br := 4.0;
fr := 0.01;

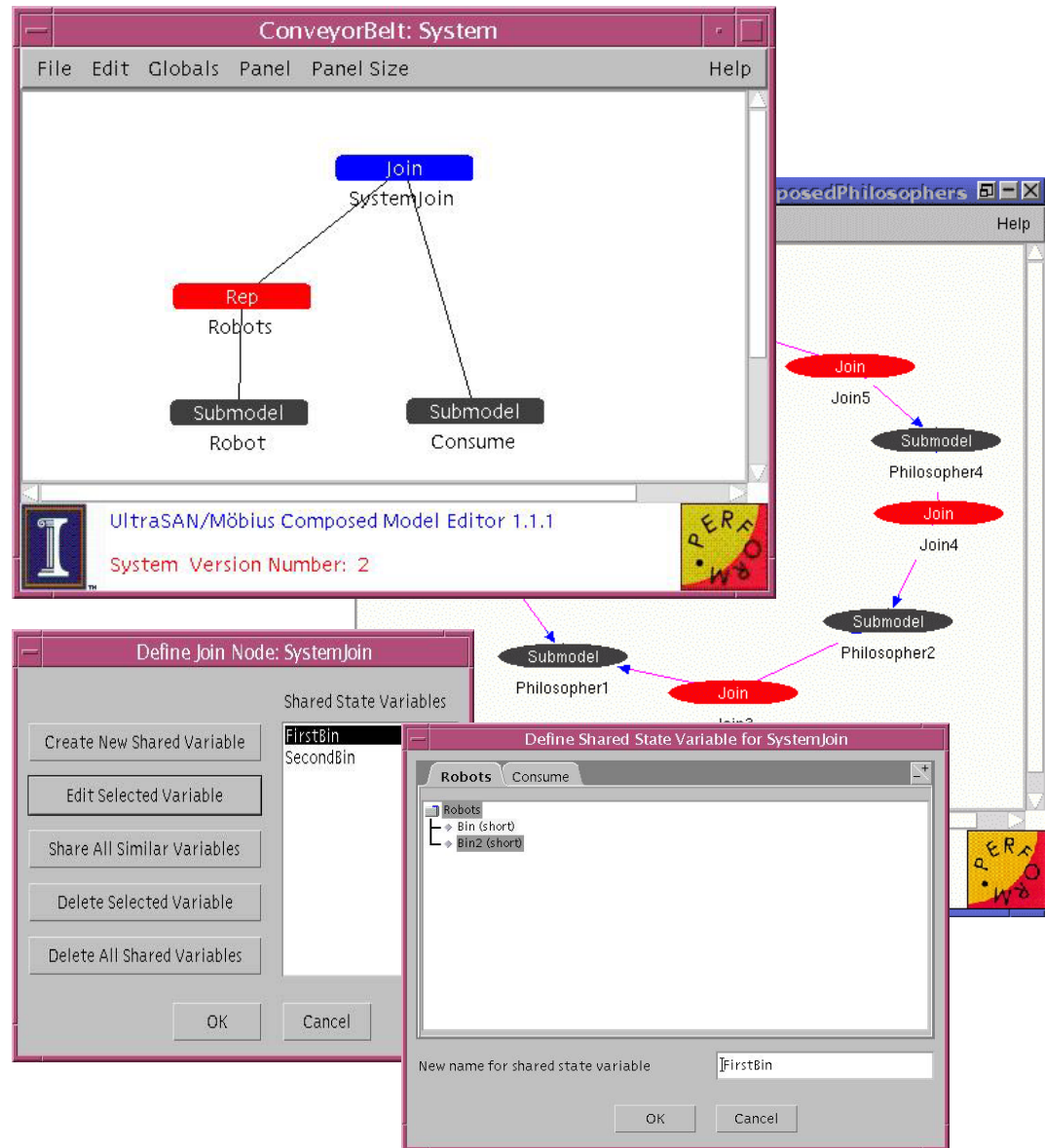
Consume[a,b] = [a > 0] => (outa, ar).Consume[a-1,b]
+ [(b > 0) && (a == 0)] => (outb, br).Consume[a,b-1];

Breakdown = (outa, T).Breakdown + (fail, fr).(recover, RecoverRate).Breakdown;

System = Consume[0,0] <outa> Breakdown;
```

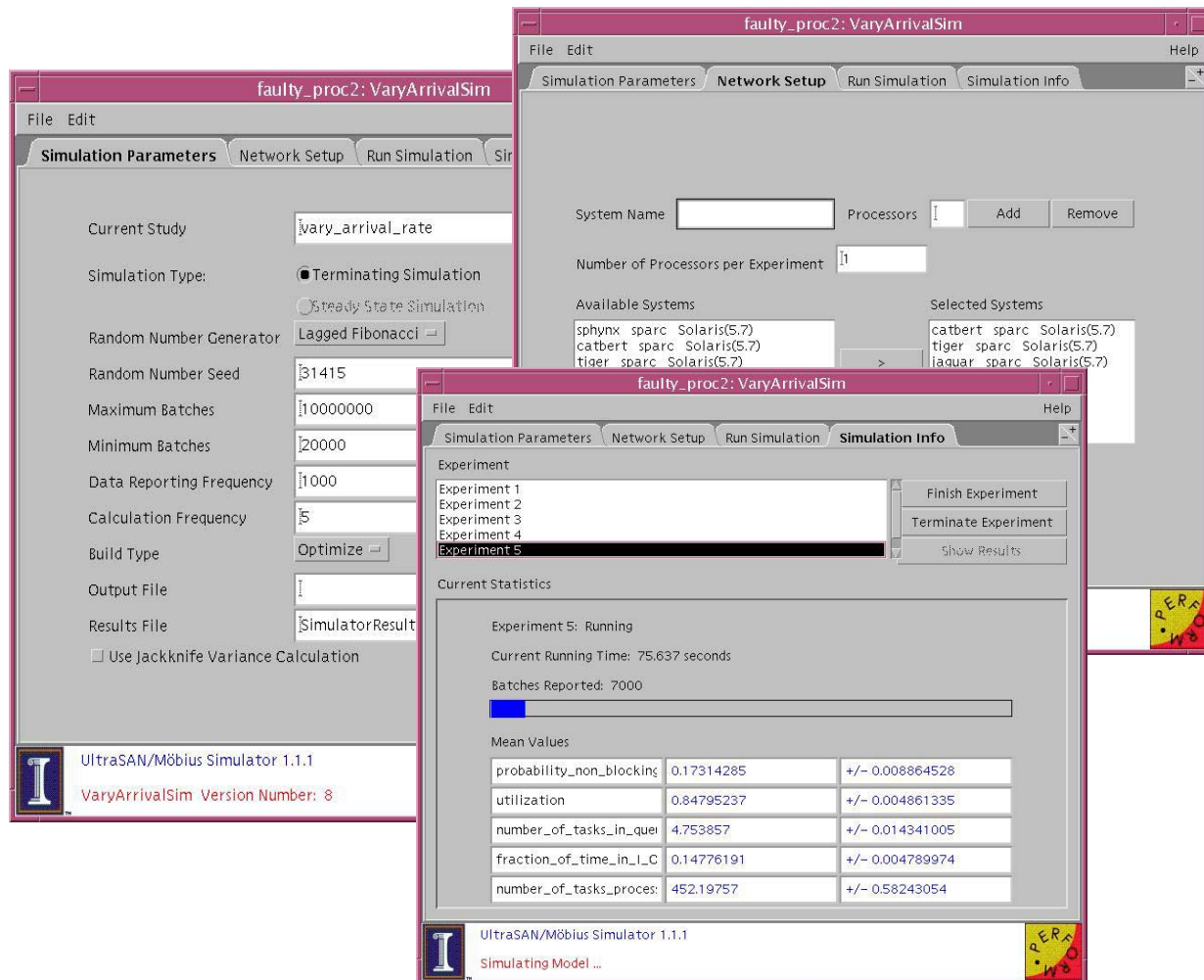
Composed Model Construction

- Composed model formalisms take models (atomic or composed) as input, and generate a model.
- Generated model implements AFI (just like an atomic model) so composed model can be further composed.
- Special properties of composition formalism (e.g. symmetry) are hidden by AFI, so can be automatically exploited by solvers and applied when only partially present in a model.
- AFI allows composed submodels to interact with one another (e.g., by sharing state variables) without understanding details of other submodel formalisms.



Simulator Use of Abstract Functional Interface

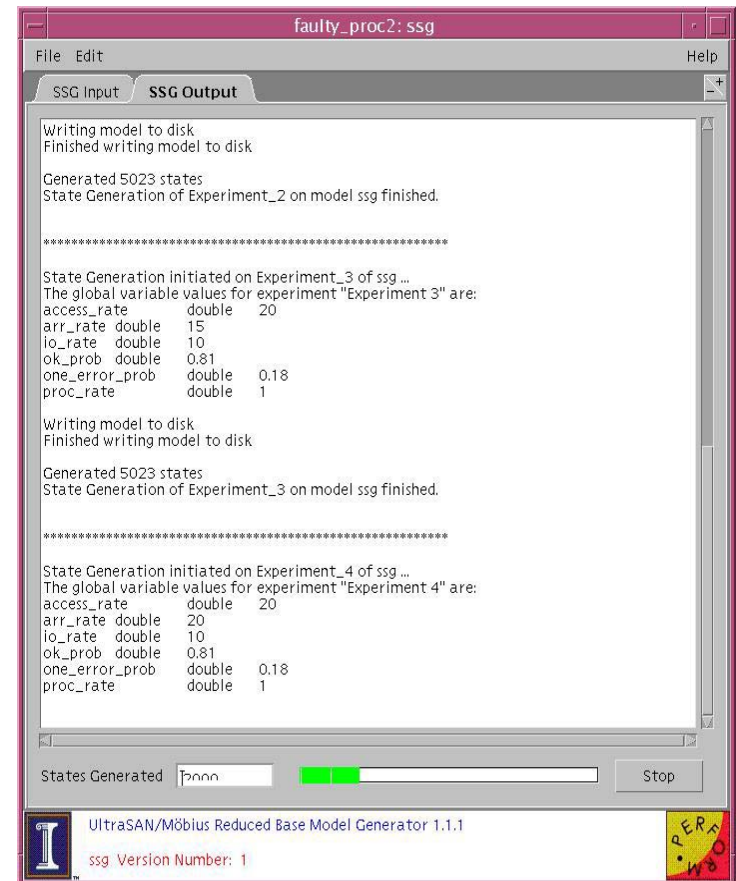
- The simulator interacts with a solvable model using abstract functional interface; Simulator does not know what formalism or formalisms it is simulating!
- Example base-class methods:
 - Base Model
 - CurrentState()
 - SetState()
 - ListOfActions()
 - Base Action
 - Enabled()
 - Fire()



State-Space Generator Use of Abstract Functional Interface

```
States =  $\emptyset$ 
NewStates = TheModel.CurrentState();
ActionSet = TheModel.ListOfActions();
While(NewStates !=  $\emptyset$ )
    TheModel.SetState(NewStates.getState())
    EnabledActions =  $\emptyset$ 
    For all Action  $\in$  ActionSet
        If(Action.Enabled())
            EnabledSet = EnabledSet + {Action}
    End For
    While(EnabledSet !=  $\emptyset$ )
        EnabledAction = EnabledSet.getAction()
        EnabledSet = EnabledSet - {EnabledAction}
        EnableAction.Fire()
        If(TheModel.CurrentState()  $\notin$  States)
            NewStates = NewStates +
                {TheModel.CurrentState()}
        End While
        States = States +
            {TheModel.CurrentState()}
    End While
```

- Generic “Fire” method changes the model’s state in a formalism-specific way
- Base class methods StateSize() and CompareState() allow state-space generator to manage state space in a generic fashion



Future Modeling Framework Research Directions

- Methods for specifying, in a user-oriented way, combined end-to-end measures of performance, dependability, survivability, and security
- Domain-specific and domain-independent modeling formalisms for computing hardware, networks, operating systems, X-load, and application software
- Methods for determining the *appropriate* level of detail and abstraction for component models, and ways to combine models expressed at different levels of abstraction (not a single, detailed, model!; models should be MEASURE-DRIVEN)
- Composition methods that use the structure of models in their solution
- Connection methods that are exact, or that give an estimate of the error they induce through their use
- Simulation/Analytic solution methods that make use of the nature of specific performance/dependability variable specifications to reduce cost of a solution
- Automated methods for collecting meaningful measurements, and incorporating measurements in the developed models
- Methods for building combined modeling/analysis/simulation/ measurement environments
- Methods for incorporating models into the network infrastructure itself, to guide adaptation (on-line modeling and simulation)