System-Level Simulation Modeling with MLDesigner

Gunar Schorcht
MLDesign Technologies Inc.
gunar@mldesigner.com

Ian Troxel
HCS Lab, University of Florida
ian@hcs.ufl.edu

Keyvan Farhangian
KVON Technologies
Keyvan_Farhangian@hotmail. com

Peter Unger
Ilmenau Technical University
Punger@gmx.de

Daniel Zinn
Ilmenau Technical University
Daniel@mldesigner.com

Colin K. Mick
MLDesign Technologies Inc
colin@mldesigner.com

Alan George
HCS Lab, University of Florida
george@hcs.ufl.edu

Horst Salzwedel
MLDesign Technologies Inc.
horst@mldesigner.com

Abstract

System-level design presents special simulation modeling challenges. System-level models address the architectural and functional performance of complex systems. Systems are decomposed into a series of interacting sub-systems. Architectures define subsystems, the interconnections between subsystems and contention for shared resources. Functions define the input and output behavior of subsystems. Mission-level studies explore system performance in the context of mission-level scenarios. This paper demonstrates a variety of complex system simulation models ranging from a mission-level, satellite-based air traffic management system to a RISC processor built with MLDesigner, a system-level design tool. All of the case studies demonstrate system-level design techniques using Discrete Event simulation.

1. The Challenge of Complex Systems

Designing, implementing and managing complex systems represents a major challenge. Complex systems range from global systems, such as. telecommunications systems, the Internet, and military combat information systems, to everyday devices (e.g. cellular telephones, computers and automobiles), to the components used to create those devices (e.g., processors, embedded systems and sensors.) Complex systems can also be processes that describe complex production or workflow systems.

Complex system design challenges include:

- Dealing with complex architectures with complex functionality in each subsystem and a high degree of concurrent processing,
- Dealing with dynamic Events with complex interaction between subsystems,
- Dealing with data, task & architecture dependent interactions, and
- Dealing with use cases and mission scenarios.

Hines [1] has described the problems of complex system design from the Military/Aerospace perspective, citing large system examples such as ships, aircraft and satellite/satellite launch systems. However, complex systems come in small packages as well. Modern processors can incorporate multiple CPU cores, onboard memory caches, and buses. Systems on a chip incorporate processors, operating systems and applications. These IC designs can contain the equivalent of millions of transistors.

As IC complexity increases and feature size shrinks, the cost of bringing the chip to market increases dramatically (see Jones [2] and Mahoney [3]) as does the potential for design errors. System–level design reduces the risk of design errors by testing the design early in the process where errors are easy to fix. Experts suggest that system-level design can determine as much as 80% of a system's total cost, performance and time to market. Accordingly, interest in system level design and analysis techniques is growing rapidly.

2. MLDesigner

MLDesigner is a unique system-level simulation modeling platform that integrates both major system-level modeling areas (architecture and function), and most simulation modeling domains, in a single tool. Modeling domains include Discrete Event, Finite State Machine, three types of Dataflow (Dynamic, Synchronous, and Boolean) and Continuous Time/Discrete Event. The domains can be used individually or used together for heterogeneous (multi-domain) models.

MLDesigner provides a complete design environment for modeling complex systems. MLDesigner can be used for a wide variety of applications including processor and computer architectural performance analysis, System-on-a-Chip (SOC) co-design (where a single MLDesigner model can represent hardware, software and operating system), wireless chip, handset and system design/analysis, and process system design and analysis.

MLDesigner models are defined graphically as hierarchical block diagrams. Blocks have defined inputs and outputs that are connected via visible links or via shared memories. Control and information is passed between blocks via particles (tokens) that consist of either a simple trigger particle or a hierarchical data structure. Bottom level blocks contain primitives written in a form of C++ code. (Source code is provided for all primitives.) Higher-level blocks contain block diagrams. All blocks can be parameterized for easy "what if" analysis and to maximize block reusability

MLDesigner libraries contain more than 2000 design blocks (operators) and more than 400 example systems. MLDesigner has a rich collection of debugging tools and dynamic control and display widgets. It is readily extensible: users can add new primitives, high level blocks, examples, new domains and links to other tools.

This paper focuses on the MLDesigner Discrete Event domain as it has the broadest scope for system-level modeling and can model the full range of electronic communications systems from applications to wave forms. By contrast, Data Flow domains are used to model DSP devices and signal modulation systems such as W-CDMA and the continuous Time/Discrete Event Domain is used to model analog and mixed signal devices such as PLLs and sensors.

The MLDesigner Discrete Event domain provides power abstraction tools: hierarchical data structures and resources. Hierarchical data structures carry functional information, protocol information, control information, costs information, and statistical information between blocks and can be used to abstract bit streams as messages or packets. Resources represent shared elements that must be acquired or accessed by an entity during a simulation. Quantity resources are "borrowed" for a while and then returned so they can be used by other model components. They are used to model entities such as buffers and memory. Server resources represent a constant flow of consumable resources such as CPU cycles.

Data structures can carry "costs" to trigger resource availability, consumption and contention. For example, a data structure representing a computer instruction can contain a field specifying the cost of the operation (in CPU cycles.) When the instruction is processed by a server resource (representing the CPU), the Server Resource automatically records the cost (for CPU utilization statistics) and calculates the delay required to represent the CPU cycles required to process the instruction. Similarly, a network packet data structure can contain a message size field (e.g., N bytes). When the message is buffered (and the buffer is represented as a quantity resource), the size of the buffer is automatically reduced by "N" bytes until the message is released from the buffer and the memory is released.

3. Example system models

This paper presents four examples showing how MLDesigner has been used to model different types of complex systems. All systems are modeled using the Discrete Event design domain of MLDesigner.

The first example is a mission-level simulation model of a satellite-based global Air Traffic Management system. This example shows MLDesigner working dynamically with another simulation tool and shows how data structures can be used to dynamically represent communications nodes (here, aircraft.)

The second example models a network of three multi-tasking computers connected in a switched Ethernet network. It shows how resources and data structures can be used to abstract CPUs, busses and cables.

The third example, is a high-level architectural performance model of a multi-processor computer, shows techniques for abstracting hardware (a computer), operating system, and application in a single high-level model.

The fourth example shows how MLDesigner can be used to develop a new application library to serve as a foundation for a whole series optical network designs and performance studies.

3.1. A mission-level Air Traffic Management system

A mission-level model of a satellite-based Air Traffic Management (ATM) system uses an imaginary network of 20 Medium Earth orbit (MEO) satellites and 11 ground stations to move control and data traffic between ground control stations and aircraft.

The ATM network carries messages for multiple services including Air Traffic Services, Airline Operation Control, Airline Administrative Communications, Airline Passenger Correspondence, Controller-Pilot Data Link, Datalink-Flight Information Services, and automated position reporting. A high-level diagram of the system is shown in Figure 1.

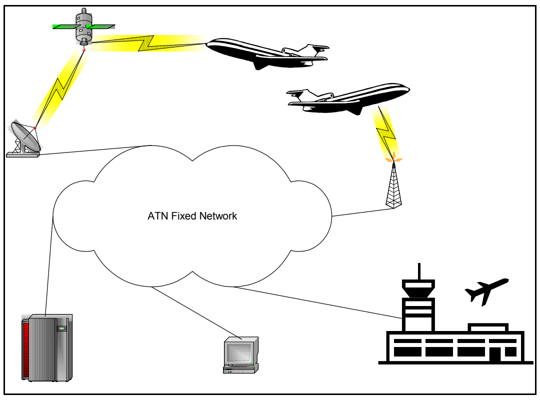
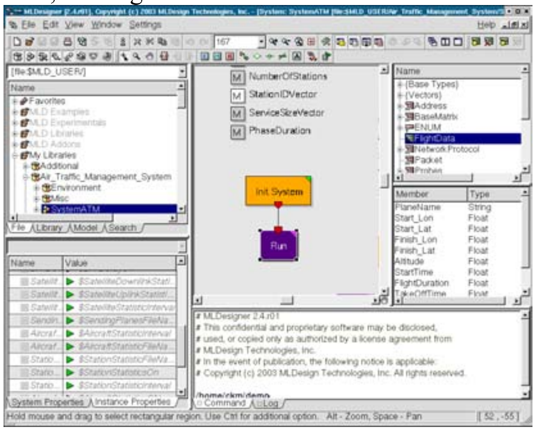


Figure 2. Schematic of the ATM System

Each message service has defined message scripts that include sender ID, message sizes and message frequencies. Flights are divided into 10 phases, each with a defined length (en-route times vary.) Each phase has a message service profile. Flights are defined with a data structure that includes flight name, departure and destination latitude and longitude, altitude, start time, flight duration, takeoff time, route sectors and reporting interval.

In this model MLDesigner works in conjunction with SatLab, a design environment for mission and system level design, animation, and analysis of wireless mobile communication and navigation systems. SatLab supplies the MLDesigner model with satellite and aircraft position information needed to calculate transmission delays. The two programs are connected dynamically via a socket interface. When MLDesigner starts, it directs SatLab to execute a predefined script, provides initialization data (e.g., time and date) and then periodically requests position data from SatLab during the simulation.

Figure 2 shows the top-level view of the ATM model in the MLDesigner workspace (center of the GUI.). Here the system is abstracted into an initialization block that loads all start-up data files, and a run module. The workspace also shows some of the top-level shared memories used to pass information between blocks. (They are the squares containing "M" at top center.) Other GUI windows show (clockwise from lower left) the model properties editor, the file manager, the data structure editor, the data structure member editor and, at the right bottom, the command window. The model properties window (lower left) is used to set parameters for the model shown in the design window. The data structure member editor (center right window) shows the some of the members/fields of the flight data structure including the name of the plane/flight, start/finish latitude and longitude, altitude, and flight start time.



Name	Value
Satellit...	$SatelliteDownlinkStati...
Satellit...	$SatelliteUplinkStatist...
Satellit...	$SatelliteStatisticInterva...
Sendin...	$SendingPlanesFileNa...
Aircraf...	$AircraftStatisticInterval
Aircraf...	$AircraftStatisticFileNa...
Statio...	$StationStatisticFileNa...
Statio...	$StationStatisticsOn
Statio...	$StationStatisticInterval

Member	Type
PlaneName	String
Start_Lon	Float
Start_Lat	Float
Finish_Lon	Float
Finish_Lat	Float
Altitude	Float
StartTime	Float
FlightDuration	Float
TakeOffTime	Float

```
# MLDesigner 2.4.r01
# This confidential and proprietary software may be disclosed,
# used, or copied only as authorized by a license agreement from
# MLDesign Technologies, Inc.
# In the event of publication, the following notice is applicable:
# Copyright (c) 2003 MLDesign Technologies, Inc. All rights reserved.

/home/ckm/demo
```

```
Hold mouse and drag to select rectangular region. Use Ctrl for additional option.  Alt - Zoom, Space - Pan     [ 52 , -55 ]
```

Figure 2. ATM model in MLDesigner

Figure 3 shows the Run module from Figure 2.

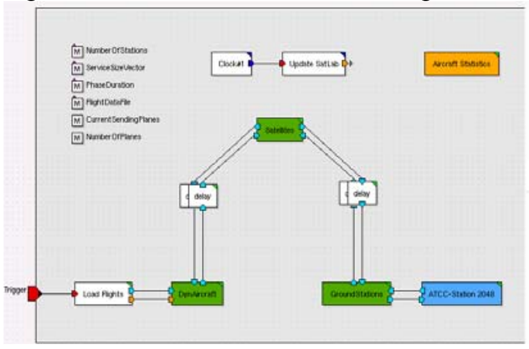


Figure 3. Top-level ATM system diagram showing top-level modules

The blocks represent top-level functions (clockwise from lower left) loading flight data, dynamic aircraft (represented with data structures), transmission delay, the satellite system, transmission delay, ground stations and a control station. Top center blocks maintain the link to SatLab; the upper right block handles statistics collection and reporting. With the exception of the SatLab clocks, all blocks shown are high-level blocks containing hierarchical block diagrams that describe their functions in greater detail. Shared memories are shown in the upper left corner.

The model generates ground and aircraft messages for each phase of the flight and sends them through the system as data structures. SatLab generates the positions for the satellite constellation and provides data for calculating the distances between the aircraft, the available satellites, and the appropriate ground stations. Built-in MLDesigner analysis blocks generate a variety of performance and behavior statistics.

3.2. A network

A small, special-purpose network (Figure 4) connects three computers through a layer 3 switch. Computers consist of CPUs connected to nodes via PCI buses. CPU1 is partitioned into two virtual machines (VM1 and VM2); VM1 executes two concurrent tasks or threads. This model demonstrates several MLDesigner DE abstraction tools.

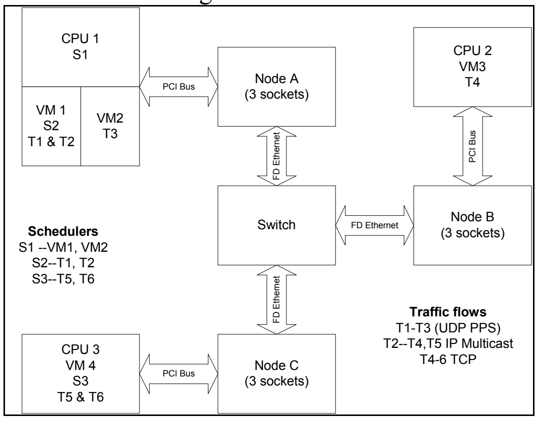


Figure 4. Schematic for network model showing virtual machines, nodes, switch and threads.

Hierarchical data structures carry messages between blocks (e.g., move routing table entries), represent the flow of data messages through the protocol stack (with support for encapsulation, addressing, etc.) and collect statistical data for performance analysis. Server resources are used to model the CPUs, PCI buses, and cables. Quantity resources are used to model memories and buffers. Shared memory makes information available to blocks simultaneously.

The top-level MLDesigner model (Figure 5) uses custom icons for the CPUs, Nodes and Switch, Server resources (the small square blocks) to model the CPUs, the busses, buffers, and the Ethernet cables. Statistical modules (the shaded blocks) collect and graph performance data.

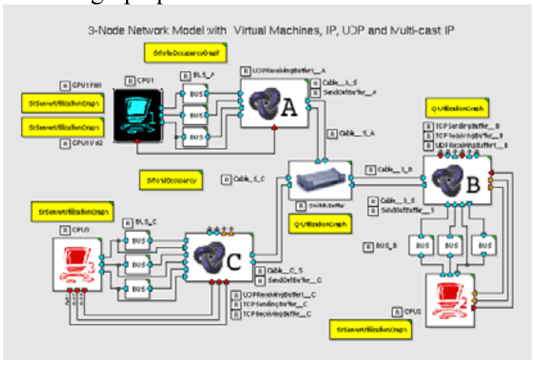

3-Node Network Model with Virtual Machines, IP, UDP and Multi-cast IP
A
B
C

Figure 5. MLDesigner top-level network model

Figure 6 shows CPU1. The top two ApplicationProcessData blocks generate messages (driven by periodic and Poisson model models) and pass them to NodeA to put on the network. The bottom Application Process Data Block receives network messages from NodeA. The two-layer server resource (lower right) uses server resources to partition CPU1 resources into two virtual machines and VM1 resources into two tasks

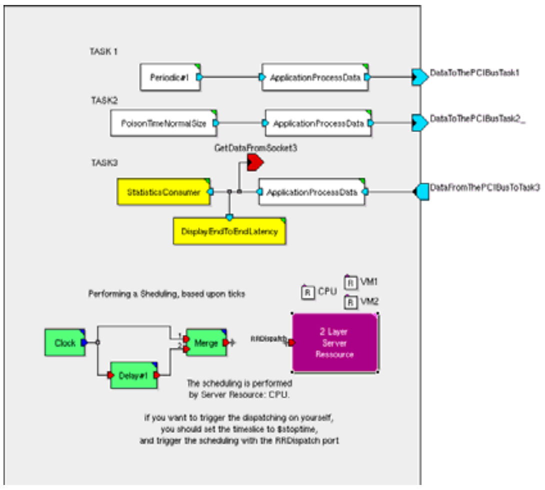


Figure 6. CPU 1 model

3.3 A multi-processor computer

This model demonstrates techniques for modeling the high-level functionality and performance of multi-processor computer architecture. The model uses independent software and hardware models that interact through a shared memory virtual connection. Parameters control key design elements such as Processor Speed, Instructions per time unit, Mean Memory Accesses Per Instruction, Cache Hit Rate, Bus Cycle Time, Number of Processors, and Memory Access Time. Probe blocks collect performance data and display it dynamically during execution or as post simulation summaries.

The software module abstracts the operating system and application as a series of processor requests (instructions) to be executed by the computer. Processor requests are modeled with a data structure. That includes fields that specify the number of instructions to be executed and timing data describing the execution of the request. Shared memories (MemoryInProcessor and MemoryOutProcessor shone in the upper right in software, and lower left in hardware) pass processor request between the hardware and software modules.

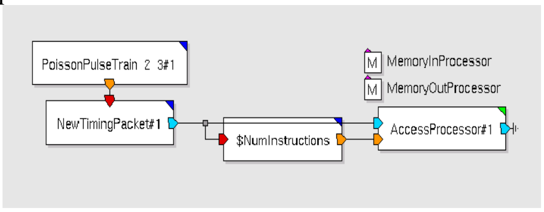


Figure 7. Software module for multi-processor computer model

The hardware model (Figure 8) has four Central Processing Units (CPUs), each with an associated Cache Memory, connected via a Bus.

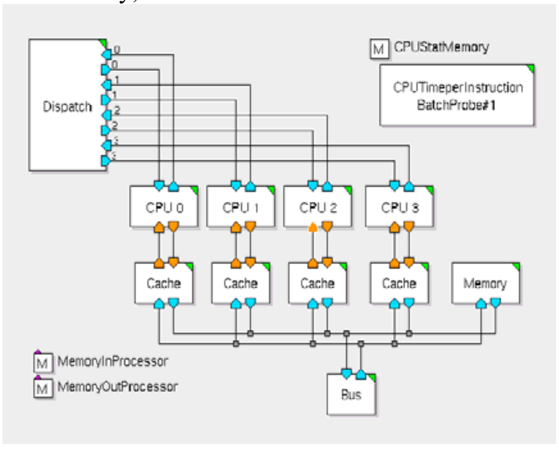


Figure 8. Top Level hardware model of a multi-processor computer

Processor requests are received by the dispatcher, which monitors CPU availability. Requests are assigned to an idle processor for execution (and the CPU marked busy), or queued until a CPU is available. When the CPU completes processing the request, it is passed back to the Dispatcher (for return to the software module) and the CPU is either given another request or marked idle. Each CPU can independently send requests to the bus for accessing the Main Memory and can send/receive requests and responses to and from I/0 devices like disk and network controllers

The CPU module decides (statistically) which instructions require Cache access, requests that number of memory accesses to the Cache module and waits for the Cache response. When the CPU module receives the Cache response, it inserts a CPU instruction delay to account for the CPU execution time for that instruction. When all the instructions for a particular CPU have been executed, the Processor Request DS is returned to the Dispatch module as described above.

The Cache decides if the memory request can be filled or requires main memory access based on the Cache hit ratio parameter. If main memory access is required, the request is passed to the bus; if not, the Cache adds a cache access delay and passes the memory request back to the CPU.

The Bus receives and queues the cache line fill or, if the bus is free, the Bus module grabs the bus and sends a request (BusDS) to the main memory. This prevents other CPUs from accessing the bus until it is released. The Bus module holds the bus until it receives a response from the main memory, applies a bus delay then s module frees the bus and returns the BusDS to the Cache.

The Memory receives cache line fill request (BusDS) from the Bus, inserts a memory access and then returns the BusDS data structure to the Bus.

All delays are inserted into processor request data structure fields and collected for statistical analysis.

3.4. An optical component library

This example shows a library of optical component building blocks to e used to develop and evaluate alternative advanced avionics network designs. Data structures are used to abstract the physical effects of optical communication components so they can be used for network-level simulation while still retaining the necessary level of accuracy. A special class of Optical Layer (OL) data structures defines all optical signals that pass through components in the optical network library. The OL class definition is used to separate optical signals from others that may be used in the simulation (e.g. electrical, wireless, logical).

Optical signals that are represented as a single wavelength within optical components (disregarding center frequency spread) are classified as members of the *Single_OL* class and are defined by characteristics such as *Wavelength*, *Opt_Power_Level*, *Data* and *Number_Of_Bits*. Optical signals of different wavelengths passing through a component at the same time are classified as wave-division multiplexed (WDM) signals and are members of the *Multiple_OL* class, which is composed of a vector of *Single_OL* members

Version 1.0a of the library is contains 38 primitive modules that perform basic functions such as laser source output and structure manipulations. The primitives are used to define 34 key optical networking components including optical couplers, splitters, amplifiers, add/drop multiplexers, filters, wavelength-tunable receivers and transmitters, and switches of various sizes.

The library's *2×4 Optical Coupler* model is shown in Figure 9. When an *OL* data structure appears on one of the component's two input ports, an input coupling loss is inserted and the structures are combined with an appropriate coupling ratio. If both input structures employ a common wavelength then an error is generated. Otherwise, an output coupling loss is assessed, the resulting WDM structure is replicated four times and the data structures are placed at each of the four output ports.

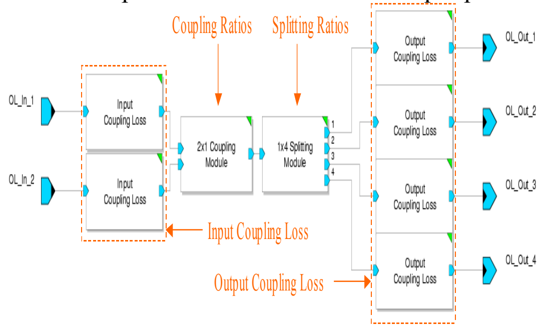


Figure 9: Optical coupler model

The library components currently model only simple physical-layer effects such as time delay and signal power-level attenuation or amplification. Future versions of the library will add component detail to distinguish between signal power and noise power so an optical signal-to-noise ratio (OSNR) can be calculated. Noise sources and effects will be added by applying several straightforward formulae [4]. Ongoing research is investigating techniques for modeling higher-order physical layer effects such as amplified spontaneous emissions, crosstalk, dispersion, temperature effects, source chirping and 4-wave mixing.

MLDesigner and the Optical Network Component Library are currently being used to investigate concept architectures for a pixel bus network. These networks consist of numerous graphics generators that create multiple display formats within the Digital Video Interface (DVI) standard. The raw, encoded data bits are sent over a network from an aircraft's Electronics Bay (EBay) to display heads in the cockpit that select the correct image and format from the streaming data. Different types of a pixel bus network models (e.g., switch-based pixel bus, WDM-based pixel bus) are being developed to analyze cost, performance and scalability tradeoffs for next-generation unified avionics networks. Future models will support TDM as well as WDM networks.

4. Summary

Four system-level models were presented to demonstrate MLDesigner Discrete Event modeling and abstraction techniques for building high-level simulation models that have a high degree of flexibility and accuracy. Each model demonstrates a different type of system and different abstraction techniques. Abstraction techniques applied include:

- Using resources to represent the costs of performing operations,
- Using data structures to support dynamic instantiation of model elements,
- Using data structures to abstract channel behavior, and
- Using data structures and resources to model the execution of software instructions and applications on hardware.

References

1. Hines, John. "We Don't Do Design Correctly." Keynote presentation from the 2001 MASCOTS conference.

2. Jones, Handel, "Analysis of the relationship between EDA expenditures and competitive positioning of IC vendors. International Business Strategies, Inc. 2002.

3. Mahoney, Jerry. "Sticker shock for photomasks." Electronic Business. 1 May, 2003.

4. Miller and E. Friedman, *Optical Communications Rules of Thumb*, McGraw-Hill, New York, NY, 2003.

Acknowledgements

Work on example 1 was sponsored by EADS/Astrium.

Work on example 4 was sponsored in part by Rockwell Collins, Inc.