Model Driven Architectures and UML Performance Modeling Capability – Design and Usage

Leonard Weinberg, Harald Pschunder, and ***Michael Stebnisky***
Lockheed Martin MS&S
Phone: 856-722-2078
Email Address: Leonard.Weinberg@lmco.com

Both the commercial and the military markets are being driven by performance requirements that far exceed the capabilities of a minimum set of computing architectures. In addition, the requirements on many newer systems are being expressed in UML, a requirements modeling language that enforces standard and consistent practices for systems and software engineering design. There is a need to justify the computing architecture designs, and to estimate the computing architecture performance for these systems. Spreadsheets and analytical methods alone are insufficient because of the statistical nature of both the messaging and the computer operating systems. This paper describes a performance modeling tool that uses an event driven design to enable evaluation of the performance of computing architectures for which the requirements are expressed in UML.

There are many computer architecture performance modeling tools on the market. However, most tools are limited in one or more of three areas: 1-a tool may operate at a very low component level making it difficult for the system engineers to use it; 2-a tool may lack a user-friendly graphical front end, making it difficult for a designer to share the modeling tool design and model results with system engineers and customers not intimately familiar with the tool; and 3-a tool may lack any compatibility with UML requirements driven methodologies. The performance modeling capability described in this paper overcomes these three limitations, while providing a robust means for estimating the suitability of UML requirements being implemented in a computing architecture.

Before we began to design the performance modeling tool, we established three goals. First, we required the ability to predict the best design for a computing architecture that achieves satisfactory performance. Second, we sought compatibility with object oriented analysis and design methods, like UML, while not precluding other approaches. Third, we wanted an open front end that would make the tool directly accessible not only to modelers, but to system engineers, software designer, and even customers.

For the last eight years, our team has been evaluating the performance of computing architectures performing critical military applications. We have been using the BONeS event driven modeling tool. In our opinion this tool far exceeded others on the market because of the low component level available which allows us to emulate computer operations. BONeS has been discontinued, and we are currently using a Lockheed Martin product called CSIM. While BONeS and CSIM both provide the lower level component capability to emulate computer operations at reasonable level of detail, these tools can also be daunting in the amount of detail that must be specified.

In order to raise the level of detail in the model design and to speed models construction, we have introduced Infrastructure/Architecture Assemblies. (This addresses model limitation Point-1 above.) An Assembly represents message flows through internet protocols, middleware, and the

components. Assemblies are chosen and connected in tandem to represent sequential message flows in a UML sequence diagram. If there are ten messages in a sequence diagram, then the appropriate ten Assemblies are chosen and connected together. Our Assembly design is a perfect match to the messages in a sequence diagram. (This addresses model limitation Point-3 above.)

In our work, four basic Assembly types have been satisfactory in representing message flows. The four types of Assemblies represent: messages entering a computer and being processed in a component, messages being processed by a component and leaving a computer, messages beginning within a computer and entering another component in the same computer to be processed, and messages entering a computer to be processed and then exiting the computer. The "personality" of each Assembly is specified by completing about ten menu-based parameters. These parameters consist of: Infrastructure/Architecture Assembly type; scenario and message information; message acknowledgement on/off; component application processing time; component application priority; node assignment; and possibly network switch port connectivity.

We estimate the time to build a model using Infrastructure/Architecture Assemblies at approximately 15% of what was typically required for model development "from scratch". A typical Assembly consists of about 40 elementary component modeling blocks and 25 default parameters settings. The design and setting the default parameters are performed one time. Each time the Assembly is instantiated typically only 10 parameters are re-set.

Recently, we have developed an export utility that extracts requirements developed using UML-based commercial tools. The extracted UML requirements information is used to support performance modeling, and introduces a user friendly interface to the model design. (This addresses model limitation Point-2 above.) We have also written a CSIM utility that makes available selected UML sequence diagram flows and generates a partially completed spreadsheet containing the architecture details for the model. UML message requirements appear in the spreadsheet. Other message attributes are added by the system engineering and computer programmers. A completed spreadsheet can be made into a static model of the system, which estimates minimum message latencies and contention-free CPU utilizations. UML requirements which we extract for the spreadsheet are: message flows present in the UML sequence diagrams; UML activity diagrams to help in selecting the appropriate sequences; and node allocation information from the UML deployment diagrams.

A critical and special feature supported by our modeling tool CSIM is the ability to build the performance model one sequence diagram at a time, each independent of the other sequences. Contention among the messages for the limited CPU resources is managed by the scheduling rules we apply to the CPU resource model. Both real time priority driven preemptive scheduling and non-preemptive time-share scheduling have been modeled.

From the point of view of the software designer, our modeling trade studies attempt to minimize the number of computing resources, while providing for long term system growth and meeting critical message latency requirements under full scenario conditions. Models are run under realistic computer program priority assignments, and use benchmark estimates for the computing platform protocol stacks, middleware and application software. One class of exciting results generated from our simulations is process timelines. These are similar to sequence diagrams but they include the message latencies due to the CPU and network contentions.