ABSTRACT

This paper presents the development of distributed simulation architecture and testbed by combining three different types of tools based on three different approaches to real-time simulation: queuing theory (with SES/workbench as supporting tool), finite state machines (with high-level software design tools, such as Rational Rose Realtime or ObjecTime Developer), and continuous simulation based on solving differential equations (injecting real-time data from a real-time simulator running on a VMEbus platform under VxWorks real-time kernel). Such a combination is very advantageous because it provides the capability of making real-time decisions in the design and simulation tools, which can help significantly improve the quality of simulations.

1. INTRODUCTION

Discrete Event Simulation is being used more and more extensively in large-scale real-time systems, which require significant number of events to be processed within a limited amount of time, including interaction with an external environment. Such areas include computer networks, telecommunication networks, distributed embedded simulation, large physics accelerators, and various types of military simulations.

The primary way of improving the quality of such complex simulations is to speed them up by distributing computations on multiple processors. Multiprocessor simulation can be divided into 2 main categories: simulation on closely coupled multiprocessor systems (parallel simulation) and geographically distributed computers (distributed simulation). Parallel and distributed computers are becoming the most powerful simulation systems as they bring together geographically distributed resources at a fraction of the usual cost.

Many of the simulation applications enumerated above accept data fed in real time from external processes to enhance the simulator’s knowledge, fix certain parameters, decrease uncertainty of models, etc. Thus, it is becoming more and more important to extend traditional Discrete Event Simulations (normally based on queuing theory), which are conducted offline, by connecting them to two different types of data sources:

- external models executing in real time, and
- true real-time processes running in the actual physical environment.

In this project, we rely on two additional types of models: finite state machines to model procedural behavior, and differential equations combined with real data. To accomplish these objectives, we combine SES/workbench with two software tools corresponding to two above mentioned data sources:

- ObjecTime Developer or Rational Rose Realtime, which is the source of real-time data from executing external models, and
- VxWorks/Tornado, which is the source of physical data from real-time processes.

2. TOOLS

In this section we briefly overview the main characteristics of the tools used: SES/workbench, ObjecTime Developer (Rational Rose Realtime) and VxWorks/Tornado.
2.1 SES/workbench

SES/workbench is a collection of software tools for specifying and evaluating network designs. It is used to evaluate the correctness and performance of computer networks. Performance is evaluated by simulating the model derived from the system specifications. Correctness is evaluated in executing the simulation by attaching “consistency constraints” that are included in every design specification. It is particularly well suited for specifying and evaluating complex systems involving a high degree of concurrent processing.

Workbench consists primarily of the following components (Fig.1):

• SES/design, which is a graphical editor used for specifying a system design
• SES/sim, the translation and simulation module for converting the design specification into an executable simulation model
• Animated Simulator (SES/scope), which provides the ability to observe and debug an executing simulation model.

Fig. 1 SES/workbench Toolkit.

SES/workbench also provides a way to interface other simulation tools at the system-level workbench model, with a facility, called COSIM [1].

2.2 ObjecTime Developer and Rose Realtime

SES/workbench simulation is based on queuing models which is not always the most optimal way to study the behavior of real-time systems. Such systems often require the use of other types of models, such as finite state machines. Modeling and simulation of an object-oriented real-time system involving a state-machine representation can be done with a software package called ObjecTime Developer [2] or Rational Rose Realtime [3]. Both tools provide features for capturing requirements, and designing, executing, debugging, and documenting designs, integrated into a complete development environment.

ObjecTime supports the creation of graphical models based on the ROOM method [4], and Rose Realtime extends this to UML notation and statecharts [5]. Using the toolset's code generation capabilities and integration with the language environments, one can create a model within the toolset and generate and link the model as an application that runs directly on the target platform.

Fig 2. ObjecTime Developer Environment.

The abstract behavior of ROOM components (actors) or UML components (objects) is described as a graphical finite state machine (statechart), which shows the allowable sequence of events that the component can process. In order to actually carry out useful activity, detailed code must be added to the states and transitions that make up the finite state machine model to perform the “actions” required, responding to the incoming events.

2.3 VxWorks/Tornado

When the simulation needs real-time data, it must be guaranteed that the data are provided on time, otherwise they may turn out to be useless. To guarantee timely processing of data needed in real-time simulations, we have to use tools that provide such capability. Real-time kernels are usually those, which make this task possible. In this project we rely on VxWorks real-time kernel with its operating environment named Tornado [6].

Tornado is an integrated environment for software cross-development, providing an efficient way to develop real-time and embedded applications with minimal intrusion on the target system. The Tornado environment comprises the following parts (Fig. 3):

• The VxWorks real-time operating system kernel, which executes on the target processor;
• A set of application-building tools (compilers, etc.) and interactive development tools;
• A full range of host-target communications options: Ethernet, serial line, etc.
The Tornado environment is designed to provide the full range of development features regardless of whether the target is resource-rich or highly resource-constrained. All of the Tornado tools are host-based, and require no additional target resources. Fast incremental downloads of application code are linked dynamically with the VxWorks operating system and are thus available for symbolic interaction with minimal delay.

Fig 3. Tornado Development Environment.

3. IMPLEMENTATION ARCHITECTURE

The target problem to be modeled with these tools involves simulation of a sophisticated hierarchical real-time data acquisition network involving bus traffic [7] with real-time sensor data supplied from external sources and workload parameters supplied by an auxiliary real-time simulation. This objective is very challenging and involves data from the synthetic aperture radar system [8] and a Kalman filter [9,10], and parameters fed from the distributed embedded simulation [11].

The first stage involved a combination of all three tools to implement a simple problem of distributed real-time simulation, which includes:
- SES/workbench doing an offline simulation
- VxWorks supplying real-time data to SES/workbench
- ObjecTime or Rose Realtime providing a user interface to the simulation and supplying parameter data to SES/workbench.

The general architecture of the distributed simulation is presented in Fig. 4. Task A1, running on VMEPowerPC/VxWorks platform is simulating a sensor. It generates a random integer every 20 milliseconds. Task A2 also running on VMEPowerPC/VxWorks is simulating another sensor. It generates a random integer in intervals, whose length is between 10 to 1000 milliseconds. Both task A1 and A2 send the data to SES/workbench for processing. Sample sensor data supplied from Kalman filter are shown in Fig. 5, both for measured and estimated values [9].

Fig 4. Five-Task Simulation Architecture.

SES/workbench accepts all the incoming data and executes the simulation in fixed time intervals, which can be changed between some minimum value and a maximum value. It is also storing the simulation results in a file. These two functions are conceptually divided into tasks B and C. Sample workbench model of the core simulation is shown in Fig. 6 [7].

Task D running in ObjecTime or Rose Realtime, executes an independent simulation and provides a simple user interface. It sends periodically to SES/workbench updates about communication traffic in ObjecTime or Rose Realtime simulation. The user can obtain the recent data from SES/workbench by typing a Display command or can terminate the

Fig 5. Sample Output of the Kalman Filter.
simulation via a Quit command. Upon a termination request, Task D will terminate all the tasks, including itself.

![Network Model](image1.png)

**Fig 6. Network Model in SES/workbench.**

For the termination to work properly, each time data is transferred, the target processor will tell the source processor whether to send again. For example, each time Task A1 sends a random number to Task B, it will get a feedback, indicating whether to send data again. All data transfers among tasks have been implemented using sockets. A sample model running in ObjecTime with calculation of message traffic parameters is shown in Fig. 7 [11].

![Code Embedded](image2.png)

**Fig 7 Code Embedded into an ObjecTime model**

### 4. EXPERIMENTS

For the 5-task configuration in Fig. 4, a number of experiments were conducted to determine the responsiveness of the individual components of the entire system, by running simple echo servers and recording the delay between sending data requests and receiving a response. All communication was implemented in sockets, using the COSIM feature of SES/workbench.

As shown in Fig. 8, response time for all experiments converges to a stable level after a significant delay. Response times for both data sources in VxWorks are substantially longer than that for ObjecTime, but this is due to network delays for communication with VxWorks. As a result we got confidence that the entire system works as anticipated and our assumptions are valid.

![Response Times](image3.png)

**Fig 8. Response Times for a 5-Task Simulation.**

The next step was to use the testbed for a more sophisticated case study consisting of a training system developed for combat vehicles [11]. Fig. 9 shows a general configuration of this system. It consists of the following five building blocks:

- Applet Interface for user input and output (not discussed in this paper) [12]
- SMC, Simulation Management and Control, which is a central part of the simulation, running within SES/workbench
- ITS, Intelligent Tutor and After Action Reporting, which runs under VxWorks
- IG, Image Generation Subsystem, also running under VxWorks
• CGF, Computer Generated Forces, simulated in ObecTime/Rose Realtime.

The performance of this system was evaluated by simulating SMC module as a server, which serves 9 clients delivering nine types of periodic messages and one client generating high-priority non-periodic traffic, according to the customer’s specification [12]. Impact of changing arrival rates on queue length was analyzed for both types of traffic. Fig. 10 shows sample measurements for periodic traffic only and three different distributions of the server population. The results show faster system saturation for slower service times, as expected.

![Fig. 10. Queue Length vs Arrival Rate for Periodic Traffic Only.](image1)

An interesting situation occurs, when non-periodic traffic is injected (Fig. 11). Arrival rates fluctuate, because high-priority non-periodic traffic is added. As soon as non-periodic traffic is settled, disturbances disappear and the curves assume shape as in Fig. 10.

![Fig. 11. Queue Length vs Arrival Rate for Periodic and Non-periodic Traffic Combined.](image2)

Finally, Fig. 12 shows how the queue length of the SMC server depends on the value of the mean service time in three cases: for periodic traffic only, for non-periodic traffic with long bound on inter-arrival time, and for non-periodic traffic with short bound on inter-arrival time. Shortening inter-arrival times has a detrimental impact on queue lengths for the same service time, but the overall loss in serviceability is not that significant, since it stabilizes at some 20%.

![Fig. 12. Server Queue Length vs Service Time.](image3)

5. CONCLUSION

Modeling of a complicated distributed embedded simulation system before the actual components are created and delivered has its known advantages, such as detection of inconsistencies in the specifications, and helps in evaluating performance of the system under design. But large-scale distributed real-time systems cannot be adequately modeled using only one tool, since each tool is based upon a particular, single underlying theory, such as queuing theory, finite state machines or differential equations. To correctly model such a complicated system a combination of theoretical principles would be required.

Moreover, to enhance the quality of the simulations done offline, it is necessary to provide real-time input data, either from live processes or from alternate independent simulations at the same time on another system. The combination of three tools, SES/workbench, ObjecTime/Rose Realtime, and VxWorks has been used to create a comprehensive heterogeneous simulation environment to meet these goals. Real-time live data from VxWorks and real-time parameter changes from ObjecTime/Rose Realtime have been used by SES/workbench to modify simulation runs accordingly.

We found out that communication among the simulation tools based on different philosophies and underlying theories is possible using SES/Workbench’s COSIM co-simulation interface. This interface allows the tools to exchange (send or get) data via the TCP/IP protocol suite. SES/Workbench can stop the simulation and change the synchronization period dynamically. Performance
evaluation is carried out based on the amount of computations performed between synchronizations.

This work constitutes an initial approach to the development of a testbed based on such an architecture, and discusses major design decisions, architectural components and first experiences in combining the offline and real-time simulations with high-level software design tools. More details can be found in [12].

ACKNOWLEDGEMENTS

HyPerformix, ObjeTime Ltd., Rational Corp., and WindRiver Systems are gratefully acknowledged for software donations, which allowed us to learn their excellent tools, educate others in using them, and apply them in our student research projects at UCF. Thanks are also due to all the support groups for donating time to answer our numerous queries.

REFERENCES