Embedded.com

Simulation-centric processes for aerospace

By Scott James, Courtesy of Embedded Systems Programming
Jan 11 2005 (14:34 PM)
URL: http://www.embedded.com/showArticle.jhtml?articleID=57700545

Big aerospace and military applications represent the extreme end of embedded-systems complexity. Here's how an embedded manager uses simulation and hardware-in-the-loop testing to break problems down to size. These guidelines can be used for any large project, or just projects that threaten to get out of hand.

In the 1990s United Defense Corporation began developing a mobile howitzer for the U.S. Army called the Crusader. The project's cancellation in 2002 prevented this impressive armament system from ever seeing battle. Crusader would have disappeared without a trace but three years later it was reborn as the Future Combat Systems' Non-Line-of-Sight Cannon (NLOS-C) land vehicle. More astonishing than Crusader's reincarnation is that the Crusader's development team is receiving as much attention for their simulation-centric development process as they are for their technological advancements.

The Crusader program was arguably the first program in the world to implement an efficient and successful end-to-end simulation-centric process. Today that process is being adopted industrywide by contractors such as General Dynamics, Boeing, EADS, and Rolls-Royce, to name a few. The simulation-centric process helps the development team manage the complexity of today's aircraft, spacecraft, land vehicles, and weapon systems by improving how the team communicates requirements. The complexity and sophistication of these platforms always increases and involves highly complex embedded systems.

At the center of the simulation-centric process is the embedded systems developer. This article explains how developers are affected by the simulation-centric process and what the various stages of the process are as they apply to a single embedded systems project. We'll discuss the timed deliverables at the various process stages including cosimulation models, real-time rapid-development units, and generational production prototypes.

Simulation-centric process

In the simulation-centric process, the team translates mechanical designs into behavioral simulation models. These simulation models are then used to develop model-based control systems. These subsystem simulations and model-based code are then used to perform real-time hardware-in-the-loop (HIL) simulation for early integration testing. Each of these stages includes a lot of iteration that might change each of the previous stages of the process. Moving design artifacts back and forth from stage to stage must be effortless. Finally, the early-integration lab is transformed into the late-stage integration facility as suppliers deliver prototypes for final acceptance testing in scheduled releases.

The mark of a highly effective simulation-centric process is collaborative design and integration during each design simulation and real-time simulation stage. The prime contractor and all the subsystem suppliers contribute simulation models for each

simulation stage. Each member also contributes to simulation result analysis and adds particular expertise to the review.

Integrated, networked nodes

Some big names in aerospace and defense, including Boeing, SAIC, General Dynamics, United Defense, Lockheed Martin, Northrop Grumman, Raytheon, Honeywell, and BAE Systems have begun implementing the U.S. Army's vision for the future of armed forces. The concept includes a force made up of highly integrated networked nodes. The nodes include both manned and unmanned land vehicles, manned and unmanned aerial vehicles, and the soldier. Information is collected independently by nodes, processed in parallel, added to a geographic database, and made available to all the nodes. Each class of node is designed to operate within a specific range of the battlefield's epicenter and to be autonomous. Thousands of different embedded systems collect data using a cornucopia of sensors, communicate over numerous wired and wireless networks, and perform a variety of embedded functions.

The expanse of technology developed during the dot-com boom will be used for fundamental shift in the capability of the armed forces. Early results have helped aerospace and defense companies come to grips with the immense requirements of testing tightly integrated embedded systems. The F/A-22 Raptor fighter aircraft has set the bar for aircraft and testing complexity. As flight-test expenditures snowballed, the U.S. Air Force implemented a new level of integration testing and simulation. Aerospace and defense companies are learning from this and other examples to design more efficient development process.

Integrated and networked embedded systems aren't a challenge only for the aerospace and defense industries. BMW received some negative press over its 745i luxury sedan and its associated recalls. When the 745i was launched, BMW proudly touted its leading-edge (bleeding-edge?) features managed by more than 80 embedded processors. Features were distributed among the many electronic control units (ECUs) that communicated with one another over several vehicle networks, including controller-area network (CAN). Numerous bugs found by end users (drivers) included unpredictable brake light operation, engine stalls and misfires, and periodic failures of the iDrive dashboard controller. Automotive companies are quickly learning the importance of efficient and reliable embedded system development processes and are willing to spend good money so they never experience BMW's pain.

Grand visions for the future of embedded systems lead to complex design, test, and implementation problems. The simulation-centric process is a summary of the successful process development strategies used by companies such as United Defense, General Dynamics, Gulfstream, Boeing, and Rolls-Royce over the years and being designed into new development programs to manage the complexity of tomorrow's integrated, networked systems.

The process in stages

The simulation-centric process approaches complex systems with the following process stages:

1. Preliminary design
2. Simulation-based design
3. Cosimulation
4. Early integration
5. Late integration

Figure 1 illustrates the process flow. Each stage in the process has one or more inputs and one or more outputs. Tools are used within each stage to work with the inputs and develop the outputs.

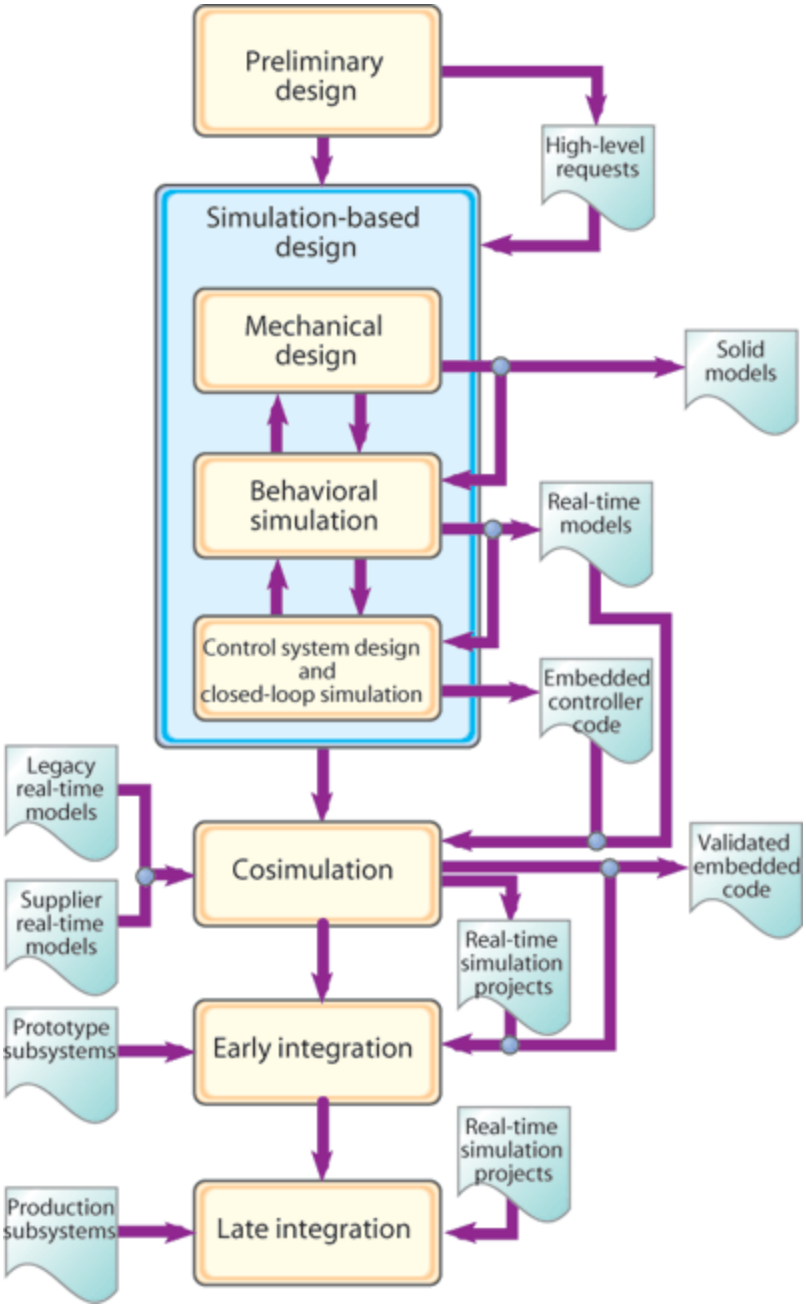

Figure 1: The simulation-centric process flow

Stage 1: Preliminary design

The simulation-centric process's preliminary design stage is similar to that of most other development processes. During preliminary design, the team defines high-level requirements for the system. These requirements may include system weight, range, speed, efficiency, and so on. Next the team breaks down the system into subsystems and specifies high-level requirements for each of those. Finally, they define the interfaces between subsystems. The preliminary design stage generates a high-level specification used to guide design efforts in the next stage of the process.

Stage 2: Simulation-based design

The second stage is simulation-based design. The goal of simulation-based design is to enable quick iterations between the mechanical design, the behavioral simulation, the control-systems design, and closed-loop simulation. First, solid models of mechanical component are developed using CAD tools. These solid models are then converted to behavioral models using behavioral-modeling tools. New features of these modeling tools allow much of this conversion to be automated. Behavioral simulations assess the mechanical design. Finally, behavior-simulation models are used for closed-loop simulation to assist control systems design and evaluation.

Simulation-based design requires a massive collaborative team effort involving mechanical designers, behavioral-simulation engineers, and control-system designers. Team members share solid models, real-time models, and problem reports. Outputs from this stage of the process include solid models delivered to fabricators, real-time models used in cosimulation and integration, and embedded control code used for early integration and delivered to embedded system suppliers. The tools used to help improve simulation-based design efficiency are evolving quickly. Table 1 lists some popular tools for simulation-based design. A tool's value in the process depends as much on its ability to import from the previous stage as it does with assisting development of outputs for the next stage.

Table 1: Stage 2: Simulation-based design

CAD Tools	Behavioral Modeling	Control System Design
CATIA	ADAMS	Simulink
Pro/Engineer	Dymola	Statemate
SolidWorks	AMESim	Rhapsody
Others	Simulink	Hand Code
	Easy5	

Stage 3: Cosimulation

The third stage, cosimulation, is a new animal for many automotive, aerospace, and defense companies. Program managers use a burden-rate calculation as a measure of the cost of using tools, machines, and facilities. A high burden rate suggests that the tool is expensive to operate. The real-time simulation computers used during early and late integration are very costly and therefore have a high burden-rate. The main purpose of cosimulation is to fully exercise all the subsystems and to validate embedded system code on a low burden rate platform (a Windows PC or Unix workstation) before prototype development or outsourcing begins.

Simulation models used during cosimulation will come from the simulation-based design stage of the process as well as from suppliers or previous design projects. The complete cosimulation will typically include many smaller subsystem cosimulation projects. Each project will be developed by the subsystem's owners. A subsystem might be owned by a team of embedded system developers, mechanical designers, or both. Cosimulation project development begins as the simulation-based design stabilizes. Execution and analysis of the cosimulation occur in parallel with simulation-based design and early integration. Design errors discovered using cosimulation will be fed back to the simulation-based design stage and will trigger a design iteration. Figure 2 illustrates how subsystem simulation and embedded code moves to cosimulation.

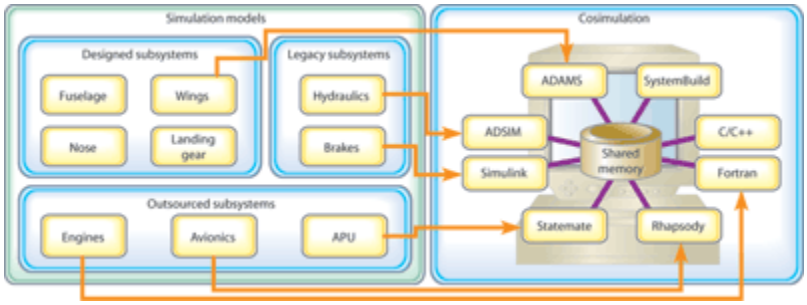

Figure 2: Moving subsystem models and embedded system code to cosimulation

View a full-size version of this image

The secondary purpose of cosimulation is to create a low-cost platform (typically a Windows or Unix workstation) for system-integration grunt work. This includes preparing models for real-time simulation and developing tests for reuse during the early and late stages of integration. The tools used for cosimulation should ideally be the same tools used in the two integration stages. Using the same tools for cosimulation and integration means the effort to configure cosimulation models, document interfaces, and create test scripts comes earlier in the development process and isn't duplicated in integration testing. The cosimulation tools must support the synchronous execution of models from numerous modeling languages. The most prevalent cosimulation tool found in the aerospace/defense industry is ADI's ADvantage/GP used by Boeing, Rolls-Royce, the U.S. Navy, Gulfstream, and others. ADvantage/GP's is popular because its cosimulation project files can be delivered to the integration stage of the process and run on ADI's PC-based and VME-based real-time simulation computers.

Output from the cosimulation stage includes validated embedded system code delivered to the embedded system supplier, projects used for real-time simulation, and test scripts that will be reused during integration.

Stage 4: Early integration

The fourth stage of the simulation-centric process is early integration. Early integration is the first step into real-time simulation and is the first time embedded system code is executed in real time. The early integration stage creates a test facility where:

1. The performance effects of network interfaces and network architecture can be tested
2. Prototype subsystems can be tested with full system complexity in-the-loop
3. Sensors selection activities can be performed (price vs. performance effects on the complete system).
4. Early human-factors testing can begin

The early integration lab is made up of numerous synchronized real-time simulators. Historically, integration labs would use one large multiprocessor real-time simulation computer to execute all the simulation models and embedded code. Experience has taught us that using many small synchronized simulators reduces real-time simulation complexity and enables more engineers to have access to a real-time computer for their initial development. Figure 3 illustrates the early integration lab for a new aircraft project using distributed real-time simulation computers.

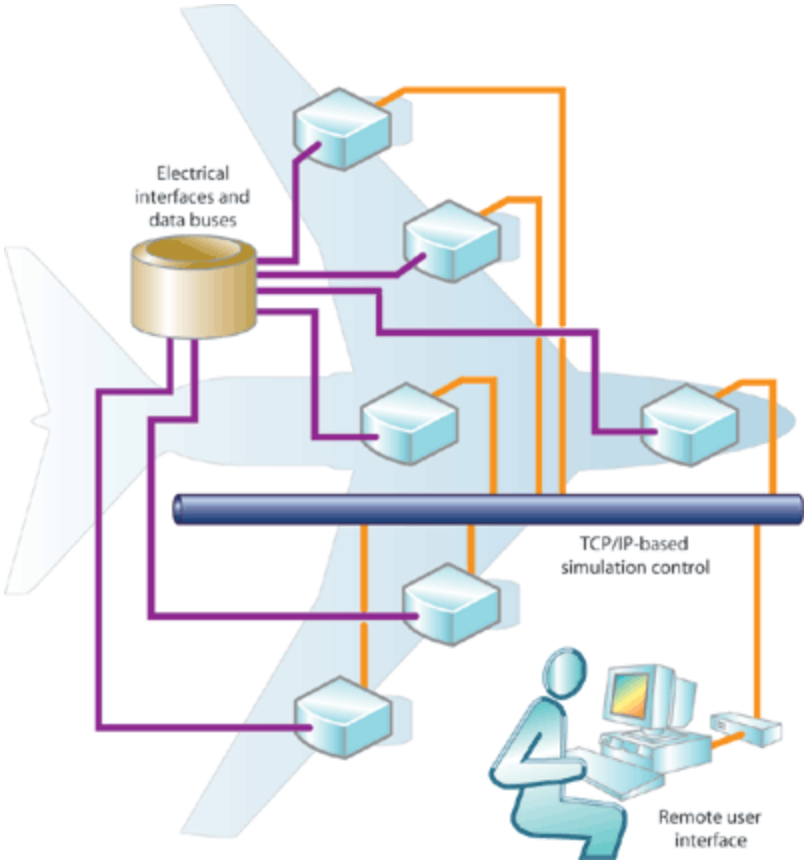

Figure 3: Moving subsystem models and embedded system code to cosimulation

Real-time systems communicate with one another using both real and simulated interfaces. Real interfaces might include analog and digital sensor signals and communication buses, such as ARINC-429, AFDX, MIL-STD-1553, CAN, and FlexRay. Simulated interfaces communicate information across non-electrical interfaces, such as the force a jet engine applies to an airframe. The simulated interface information is communicated using high-speed communication hardware, such as gigabit Ethernet, reflective memory, or a Scramnet fiber optic interface board. The distributed real-time systems must use a common clock between them to allow test data from one system to be compared accurately to that of another system.

When dealing with networked embedded systems, a problem in one subsystem's simulation might originate in a different subsystem. A common clock enables post-run analysis to find the source of the problem. The common clock is added to distributed real-time systems using an external clock such as IRIG-B. Each real-time simulation is controlled from a remote user interface running on a PC or Unix workstation. In addition to control, the remote interface provides visibility inside simulation models and embedded code for the purpose of analysis and debugging.

Stage 5: Late integration

The fifth and final stage of the simulation-centric process is late-integration testing. Once early integration is complete, the early integration facility is transformed, subsystem by subsystem, into the late integration facility. As subsystems arrive from suppliers and development teams they're scheduled as late-integration activities. Subsystems are added one at a time to isolate the source of any problems. Suppliers might schedule the delivery of several generations of prototypes over a period of perhaps a year. Early generations are modular and have easy access to circuits for probing. Swappable daughterboards allow different microprocessors and sensor circuits to be accurately evaluated. Later-generation prototypes begin to look like the final production system. Circuits become more dense and harder to access. Cooling, structural rigidity, and vibration damping are all accounted for in the last generations.

The sources of many embedded software problems are modern cockpit displays and their interaction with avionics and other control computers. The late integration lab will typically go through dozens, and even hundreds, of software upgrades before the first field/flight test on a new system. Test pilots will spend thousands of hours evaluating and testing complex controls and displays in the late-integration facility. Late-integration facilities are often very expensive because they include millions of dollars of prototype equipment. The high burden rate of late-integration labs requires that all possible test development occurs during early integration rather than late integration.

Developing for embedded

Many aerospace, defense, and automotive companies in recent years have begun feeling as though they've lost control of their embedded systems. For example, the vast majority of new features added to an aircraft are implemented in an embedded system rather than through advancements in mechanical design or construction materials. Avionics computers are an aircraft's brain and as such all new embedded systems in an aircraft will touch the avionics computer. Many aircraft manufacturers outsource the complete avionics computer system. As a result, the aircraft integrator is at the mercy of its avionics supplier to provide new features in a timely and reliable manner. As more and more of the cost of an aircraft is in the embedded systems, the aircraft integrator has control over less and less of its own product.

Many aerospace, defense, and automotive OEMs are realizing they are in the business of software development and are using the simulation-centric process to regain control of their embedded systems. By embracing model-based control-systems development, OEMs are able to keep application knowledge in-house and only outsource a "dumb" embedded system. The so-called "operating system/application system separation" is used by aircraft embedded system suppliers such as Hamilton-Sundstrand, Goodrich, and Honeywell to describe the outsourcing relationship. In this context, operating system (OS) means the physical system hardware, a real-time operating system, or scheduler, device drivers, and a well-documented API. The application system (AS) includes all the application code running on the OS. The AS is developed in-house; the OS is outsourced.

An embedded systems developer may find himself on either side of the OS/AS relationship. Regardless of which side you inhabit, you'll be asked to make deliveries during the cosimulation stage, the early-integration stage, and the late-integration stage of the simulation-centric process. Communication is improved and the blame game is eliminated by getting the responsible technical experts in one test facility at the right time and completing a single test matrix. As problems are found they're characterized and isolated. Once bugs are isolated the owner is identified. Iterations are made, embedded software bugs are eliminated, and development programs are a success.

Scott James is the director of business development at Applied Dynamics International in Ann Arbor, Michigan. Scott has managed development projects for companies including Rolls-Royce, Gulfstream, Boeing, United Defense and General Dynamics. sjames@adi.com.