Introduction

Building complex real-time control systems is a huge challenge. Sophisticated control algorithms must be carefully designed, simulated, tested, and then embedded into complex software that coordinates interactions with many other systems. The special needs of real-time systems—sequencing, incremental testing, online operator interfacing, and live debugging—compound the problem. Engineers, software programmers, managers, and test personnel must cooperate to design, develop, and test the system. The development spans a long, complex process, from initial prototyping to final system implementation and testing. It’s a true “end-to-end” challenge.

Of course, many types of systems require team effort. Complex control-system teams, however, face a unique challenge: integrating sophisticated behavioral designs into a complex software system. This is quite difficult. In most organizations, different groups, acting nearly independently, develop the controls and the software. Integrating these two camps has become one of the great barriers to the implementation of aerospace, process control, and other complex systems. Why is this? What makes it so difficult to integrate control designs into well-written, complex software systems? To answer these questions, we must first understand both the challenges and the teams.

Challenges and Teams

Control-algorithm designers construct dynamic system models, develop algorithms, and simulate the proposed designs. Control engineers prefer a straightforward, concrete approach. Control engineering tools speak the mathematical language of numeric algorithms and provide fast modeling and numeric simulation.

Software developers, however, provide the infrastructure that glues the system together. They develop specialized modules, such as network communication, device drivers, and user interfaces. Their primary challenges are abstract structuring and practical system organization. Software developers use programming tools that provide coding flexibility, architectural design, code management, debugging, and reuse. Their design tools use abstract techniques that emphasize object-oriented design and structure.

Controls engineers are not usually trained in the advanced techniques of software development, and software developers rarely have the engineering background for control-systems design. Starting from scratch with abstract modeling languages, the “modern” software design often ends up poorly adapted to control. The path from a blank-sheet software architecture to a working control system is simply too long to succeed.

The Problem

Controls engineers develop the algorithms and then “toss the design over the fence” to the development team. The software team rewrites all the algorithms from scratch, just to fit the computational structure. The project takes forever. System integration takes longer than forever. Rework, such as revising the control algorithms, is painful. It’s hard to reuse code, algorithms, or even the framework. The whole design remains in the dark ages.

So what can we do? How can complex software systems incorporate sophisticated control designs?

The Solution

The solution is simple: provide tools that work the way the teams think. Let control engineers develop with state-of-the-art dynamic modeling and algorithm design tools. Then integrate the control system design into a targeted,
proven software architecture with complete system-development tools to accelerate implementation and testing. The result will be an end-to-end tool chain that satisfies both engineers and programmers.

RTI has provided the first true end-to-end tool chain by integrating Simulink with ControlShell. Simulink, by The MathWorks, is the tool for developing and simulating control systems. Simulink excels at dynamic systems modeling. It provides a powerful modeling and simulation environment. With MATLAB, Simulink also provides detailed controls design and analysis.

ControlShell is an application framework designed explicitly for software development of large control systems. It deals with complex and intelligent subsystems, wraps legacy code, and easily incorporates complex, custom modules. It can handle large applications developed by teams of programmers, engineers, architects, and managers. ControlShell is a clean, object-oriented design that works in the real world.

Together, ControlShell and Simulink address both controls and complex system software. They take the design team all the way from systems prototyping and controls analysis through simulation and test to final application code. ControlShell and Simulink form the end-to-end tool chain that streamlines complex system development.

This satellite simulator application was developed in ControlShell. It includes a Simulink model of the satellite dynamics. The Simulink model is simple to integrate into the software design.
Simulink and ControlShell

Simulink is a dynamic-system modeling tool. ControlShell is an application framework and software-development environment for dynamic systems. While they have superficial similarities, they really address different problems. Simulink excels at simulating physical systems and developing control algorithms, while ControlShell excels at developing and managing all the software for complex systems that include control subsystems. Together, Simulink and ControlShell enable complex system development.

Let's examine them a little more closely…

Simulink

Simulink is a software package for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Simulink also supports multi-rate systems, i.e., systems that have different parts updated at different frequencies.

For modeling, Simulink provides a graphical user interface (GUI) for building models as block diagrams, using click-and-drag mouse operations. With this interface, you can draw control block diagrams just as you would with pencil and paper (or as most textbooks depict them). This is vastly superior to previous simulation packages that required you to formulate differential equations and difference equations in a language or program. Simulink includes a comprehensive block library of sinks, sources, linear and nonlinear components, mathematical functions, and connectors. You can also customize and create your own blocks.

The Simulink environment also includes an optional state-diagram editor known as Stateflow. Stateflow describes logic sequencing in control systems, and supports flow-diagram notation for common code structures such as for loops and if-then-else constructs. You can use flow diagram notation and state transition notation seamlessly in the same Stateflow diagram.

Simulink is a superb simulation tool. After you define a model, you can simulate it with a choice of high-fidelity integration methods. Simulink includes model analysis tools, including linearization and trimming tools. Simulink supports both batch and interactive simulation. Batch simulation uses MATLAB's command-line interface; it's very useful for Monte Carlo simulations or parameter sweeps. In interactive mode, you can see the simulation results while the simulation is running. In addition, you can change parameters and immediately see what happens, for easy “what if” exploration. You can also export the simulation results to the MATLAB workspace for post-processing and visualization.

Finally, Simulink has a code-generation option called the Real-Time Workshop (RTW). With RTW, you can generate code from your diagrams, compile it, and download it to run on various real-time targets. All told, Simulink/Stateflow/RTW is a well-designed, complete controls development environment. It's elegant, powerful, and amazingly useful.

ControlShell

Once the control system has been designed in Simulink, an enormous software development process still remains. It includes elements such as network communication, device drivers, external user interfaces, legacy code, and interfaces to other design tools. ControlShell manages this process; it is a complete software development environment, specifically designed for complex, intelligent control systems. It encompasses all the tools and ‘glue’ required to build applications.

ControlShell includes an application framework that specifically targets control and simulation systems. An application framework is a combination of pre-written software that accelerates application development. It includes an object hierarchy, a pre-defined structural framework, and pre-written libraries that form the basic program structure. Teams use the framework to coordinate their efforts, build and test software and hardware, and deliver a working application. If you select the right framework for your application, you start the project halfway done.

Using a pre-built, proven structure is certainly important. However, using an application framework provides other benefits as well. For instance, the common structure enables reusable software components.
ControlShell is built around an integrated component repository. Team members use the repositories to share work. Developers write new components and store them in the repository. Engineers put together systems built from components in the repositories. You can design and program complex, intelligent control applications using reusable software components. You select components from the component repository, connect them with graphical diagrams, build, download, test, and deploy, all within a single, integrated development tool. The repository-centric view of the world provides a reuse mindset that pervades the entire project. The real power of an off-the-shelf framework is that using an accepted structure lets you leverage targeted tools and powerful run-time services.

The ControlShell environment integrates many tools, including graphical tools for data monitoring, performance testing, memory analysis, source-level debugging, function/method tracing, live state-machine animation, scenario capture and replay, and test scripting. ControlShell has a complete compilation and build system, code and diagram level debugging, and a customizable integration with revision-control software tools. It's a superb programming environment, designed for run-time test and verification. You can peek and poke at any component, data, or logic in the running system. All the same run-time facilities are available with any target platform, whether it's a local Windows or Unix simulation or a remote real-time application. This makes the task of debugging, testing and integrating the system much easier and faster.

**ControlShell and Simulink Work Well Together**

RTI has closely integrated ControlShell with Simulink and RTW. Simulink diagrams coexist as blocks in ControlShell diagrams. Most Simulink diagrams are fully integrated, including discrete and continuous dynamic blocks and Stateflow diagrams. To edit a Simulink diagram, just open it from ControlShell. Changes to the Simulink block, such as changes to the number and type of the input and output ports, automatically flow through to the ControlShell component without any user coding. Many Simulink diagrams, and even multiple instances of the same diagram, can be included in a single ControlShell system.

With this tight integration, you can easily use Simulink to develop your physical models, prototype the system and develop your control algorithms. You can then include those models in any ControlShell system. The integration is seamless; you use each tool for what it does best.

**A Typical Process**

ControlShell and Simulink fit naturally into your team's process. They manage each of the steps, from system analysis and prototyping through test and integration to maintenance and upgrades.

ControlShell's top-down, bottom-up methodology makes it easy to combine complex software with sophisticated controls. You start development with a single block that represents the entire system. Next, you break the system—from the top down—into functional or physical subsystems. When the functionality of subsystems is clear, you build the subsystems—bottom-up—from reusable components. The components can be diagrams, sophisticated custom code, or Simulink diagrams. The single ControlShell IDE makes all these easy to access graphically.

Let's examine the stages of a typical development process for a large aerospace system, and see how ControlShell and Simulink help.

**Prototyping and Requirements Capture**

Simulink makes it easy to test and validate an accurate model of the system. Using the collection of linear and nonlinear components, you can build and simulate models of plant, sensor, and actuator dynamics. With ControlShell, the prototype can include the overall software structure, including interfaces between major functional modules. Many systems can be prototyped quickly by reusing previous code modules or selecting from RTI's libraries.

**Systems Analysis and Design**

Once the prototype is complete, most large projects start with systems analysis. Working from high-level requirements, systems engineering breaks the problem down into manageable pieces. During this process, ControlShell allows you to develop well-specified interfaces between the subsystems. You then use the formally-
defined interfaces to communicate the design to the other teams. The handoff is graphical and seamless. Better yet, all interfaces are actual code specifications that can be enforced, verified, and maintained.

**Control Design**
The controls team will use Simulink/MATLAB to design the control algorithms. With the plant model, you can test your closed-loop system in simulation to determine how well the design meets performance requirements.

Once you have a Simulink model that you’re satisfied with, it is a simple matter to incorporate the model into your ControlShell diagram. You generate C code directly from the Simulink block diagram using RTW, and the code is automatically integrated into your ControlShell application as a component. Multiple models are easy to support; just hook them up on the ControlShell diagram.

**Coding**
Once the system design is available, the teams need to implement the software for the subsystems. The various teams can use Simulink and ControlShell to develop the code. For instance, you can develop Simulink/ControlShell components to communicate over a network or some other communications channel (like telemetry and command on a satellite). You can write device drivers to interface to the hardware. You can also easily include legacy code and interface to pre-existing subsystems. The well-defined interfaces help ensure that everything works together.

**Simulation and Testing**
You can, of course, easily simulate the Simulink-designed components. Then, when you are ready to test with the full application, simply treat the model as a component within ControlShell.

The larger system is also easy to simulate. ControlShell supports real-time, hardware-in-the-loop simulation with “mix and match” software and hardware subsystems. You can easily simulate with a software model of a sensor implemented as a Simulink plant today, then swap in a device driver implemented as a ControlShell component and test against the actual hardware tomorrow.

Of course, the rich testing and debugging environment is critical at this stage.

**Integration, Delivery, Maintenance**
Finally, you need to deploy the software on the deliverable hardware and do final testing and integration. When you deliver the system, ControlShell’s tools help you install and debug the application. All the debugging tools available in the development environment, including the command-line shell, data monitoring, and state-machine tracking, are available on the running system. You can use your familiar environment to iron out the last-minute site-integration bugs. The tools can even be used remotely, so you can help debug from the home office.

ControlShell also makes it easy to maintain and upgrade the system in the field. To change a piece of the code, just swap out a component. The debug tools are always available. Maintenance is much easier with assurance that the actual system matches the ControlShell graphical design and automatic documentation.

**Conclusions**
ControlShell is a complete software integration and development environment. It excels at coordinating the efforts of teams developing complex software systems that include control systems.

Simulink is a high-fidelity control design tool. Simulink is the best tool for simulating physical subsystems and developing control algorithms and logic.

Together, ControlShell and Simulink bridge the gap between complex software and sophisticated controls. They let teams design, create, and deploy complete solutions faster. They combine best-of-class solutions to both complex systems software and controls mathematics. They form the end-to-end tool chain that can help engineers and programmers work together.