

CommsDesign

Getting Graphical in 3G Wireless Designs

With pressures mounting in the wireless sector, 3G wireless handset and base station designers need to mitigate all design risks. New graphical modeling techniques can help out.

By Mike Woodward, RadioScope

The stakes are rising in the wireless design community. Now that carriers have spent tons of cash on 3G licenses, base station and mobile developers must deliver systems now that will make 3G networks come to life. Miss a market window and your company may be out of the race all together.

The key is to reduce development risks. With that in mind, designers should consider moving away from traditional modeling techniques toward newer graphical modeling approaches.

In this article, we'll compare current modeling techniques with newer graphical modeling techniques. We'll also examine how graphical modeling can be implemented and its impact on the overall system design process.

Modeling the Traditional Way

Modern software development uses tools such as the UML, but still focuses on performing most of the design work at the start of the project. To focus the discussion, imagine being asked to develop a simple model to investigate a simple 3G-style transmit-receive structure consisting of a single user, convolutional coding, spreading, and scrambling. A fairly reasonable UML model for this system is shown in Figure 1.

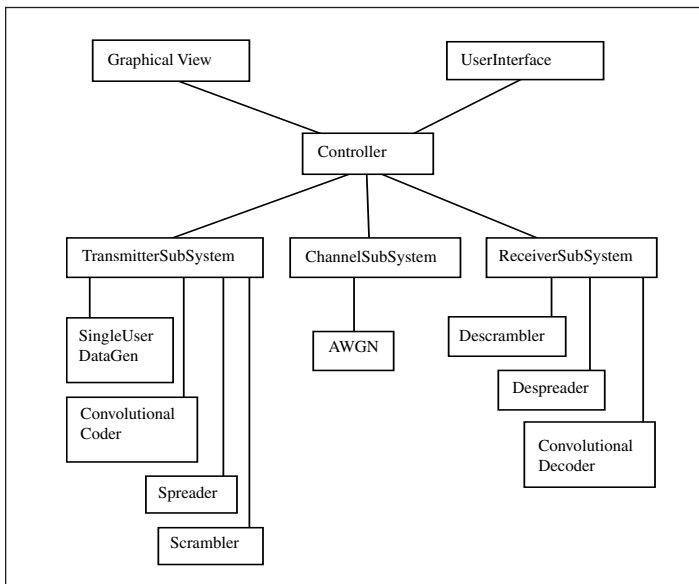


Figure 1: Example UML design for a simple 3G model.

One immediate problem is that there is no implied ordering of the operations. If a user wanted to understand the architecture being used, it is not sufficient to look at the underlying UML model since will not tell the developer the order of, for example, spreading and scrambling.

Time is another problem here. Using the UML modeling approach, designers must spend time writing graphical display code and building a model control infrastructure.

The situation gets worse, however, when the designer tries to extend and expand the model. For example, the addition of a turbo coder would mean changes to the transmitter, receiver, control, user interface, and graphical view. Even worse, it takes a re-compilation to look at different receiver architectures.

The Graphical Approach

Graphical modeling offers the software modeler a better way to move forward. Graphical modeling is a way of combining the underlying code and design in a single artifact. We can define graphical modeling as follows. The code that executes the block functions is written to a standard interface. The graphical modeling environment manages the connections and data flows between the blocks. Finally, the system maintains a graphical representation of the underlying connections. These connections are presented to the user visually.

There are a number of key differences here from the traditional

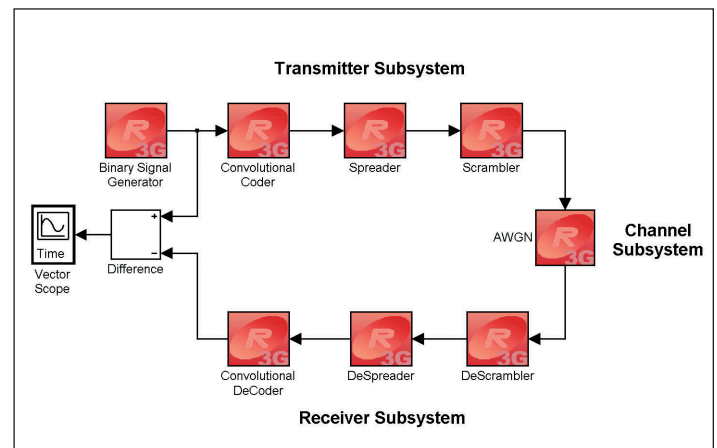


Figure 2: The simple system of Figure 1 implemented in a graphical environment through Simulink.

approach to simulations. First, unlike traditional tools, the user interacts with the graphical representation. Second, since the design is the model and the graphical modeling blocks have a standards interface, the design of a system can be changed at any time prior to running without recompilation.

Figure 2 implements the same 3G design from above using a graphical modeling method. Figure 2 can be considered an executable design. Each block performs some action, and there is code 'underneath' the blocks. The lines between blocks represent data flow, one line per item of information. For example, a vector of 2560 double precision numbers is represented by one line and a single integer number is also represented by one line. This notion is very similar to arguments in C functions, one item of information per argument.

As Figure 3 highlights, every port in the graphical modeling environment corresponds to a data item in a routine declaration and the lines between blocks represent the data flow. The graphical environment takes care of the actual data flow between blocks, so blocks can be connected in any valid order.

The code 'underneath' the blocks is compiled into a Windows dll (for Windows-based machines) or other formats for other operating systems. When the graphical model is loaded into the graphical modeling environment, the underlying dlls from the blocks are loaded into memory. When the model runs, the graphical modeling environment calls the blocks in turn, supplying them with their input and getting their output. The output from a block is passed as the input to the next block in the chain.

So we can think of graphical modeling as just a chain of DLLs with the linkage between them managed by the environment. We can view graphical models, like the one displayed in Figure 2, as showing the helpful information to the use but also as instructions to the modeling environment about how the data is to flow between DLLs.

Advantages of Going Graphical

The graphical modeling approach described above, provides some

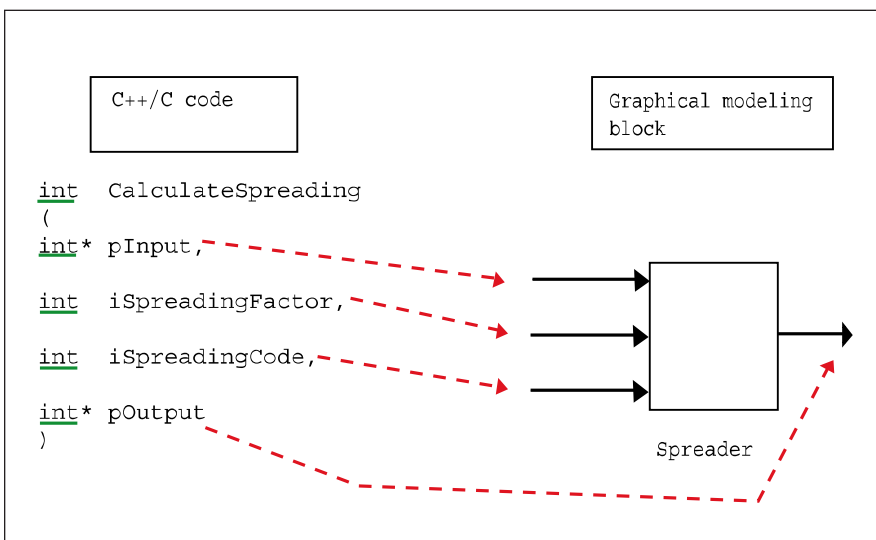


Figure 3: C++/C routine input and output compared with the equivalent graphical modeling input and output.

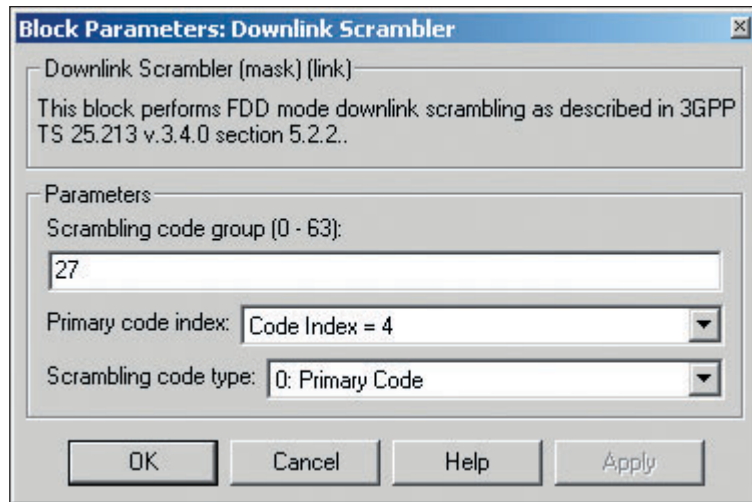


Figure 4: Through Simulink, designers can enter control information for a block via a dialog box..

nice benefits to design engineers. Under this modeling approach, blocks can be swapped, added, or deleted without compiling a single line of code. Blocks from third parties can also be added, or the designer can add his own code easily, provided it conforms to the environment API. For example, Figure 2 shows an off-the-shelf third-party component being used to display the data (the Vector Scope), saving time on writing a graphics block.

Under the graphical modeling scheme, blocks also exist to send data to disk or to read it from disk. This means that a developer is spending more time on writing the algorithms and on designing their system, rather than on writing non-core code.

Since the graphical environment here is managed, the developer can split data lines and send the output from one block to two or more blocks. Designers can also swap or add more blocks. For example, additional graphical output blocks can be used at any point in the model.

What about the model control? For the purposes of this discussion, we're using Simulink to display the graphical model. In Simulink, general control features are controlled via the Simulink interface, for example the simulation time step and the length of the simulation.

Block specific items, on the other hand, can be controlled in one of two ways, via a simple dialog box or via data sent to the ports directly. Figure 3 above shows a case where the control information is sent in via ports (the spreading factor and spreading code). On the contrary, Figure 4 shows a typical dialog box (Note: This is the dialog box used in the spreader in Figure 1). A dialog box like the one displayed in Figure 4 can be constructed in less than five minutes using the tools within Simulink.

Development Using Graphical Modeling

Above we laid out the how the graphical modeling works. Now let's look at how graphical modeling differs from a practical standpoint. To do this, we'll look at the common stages of model development, namely design, coding, and debugging. We'll also look at a feature unique to graphical modeling.

The graphical modeling technique completely changes a designer's modeling process. Designers do not need to create an infrastructure as the graphical modeling environment provides the infrastructure. Since the graphical model is the design, UML designs are no longer required and the design decision comes down to deciding how many algorithms should be coded into each graphical modeling block.

For example, there should be separate spreading and scrambling blocks rather than a monolithic transmitter block. Because reuse is strongly enabled by graphical modeling, we can use existing graphical blocks or even blocks for AWGN, multipath, or turbo decoding. This leaves only the blocks that the developer needs or wants to develop. For each of the algorithms, the developer needs to decide what inputs and outputs the algorithm block should have. This is not much different from deciding what arguments a C function should have, and for most algorithms it will be immediately obvious. The design stage has just shortened dramatically.

Under the graphical modeling approach writing code changes only a little bit. It comes down to filling in sections of an API. Obviously an API needs to be learned, as well as how the environment is constructed. These are just one-time costs. Most graphical modeling environments ship with example code to allow the developer to get up and running quickly.

The new step in this process is creating the block in the graphical modeling environment and linking it to the underlying dll. If a designer is using Simulink, this is very straightforward and can be achieved in a minute or two. It consists of little more than a simple GUI construction and entering the name of the underlying dll. Once again most environments provide examples.

One concern that surrounds graphical modeling is whether debugging is substantially different. This is not necessarily the case. Most environments offer a graphical debugger at the block level. However, most tools also support tried and tested C or C++ debuggers

Graphical Modeling Problems

Clearly, graphical modeling is not a perfect solution. It also has some problems.

Overhead is clearly one of the big concerns surrounding the graphical modeling approach. There is an overhead compared to performing the same calculation in a straightforward C or C++ model. However, if the code is written in C or C++ and compiled into a dll this overhead is small. Bear in mind that the graphical design environment doesn't affect the speed of execution as the connections between blocks, and their positions, don't change during simulations. The overhead comes down to the environment passing data between blocks.

Speed can also be headache. To be frank, some graphical models are slow. But this has more to do with how they are written. It is possible to write blocks in the built-in interpreted language, but this will always be slower than compiled C, and in some cases a lot slower. A better option is to build blocks in C/C++ and then convert them into dlls. This will accelerate the graphical modeling process.

Graphical Modeling Wins

Graphical modeling offers much more rapid deployment and a faster time-to-simulation than writing C or C++ models directly. This comes out of the reduced design time, the managed environment and the availability of built in graphing and third party blocks etc. Once built, graphical models are more flexible and different models can be constructed without having to re-compile code. The cost to doing simulation this way is a shift in thinking, learning the API and a low run-time overhead.

Author's Note: For more information on Simulink, visit www.mathworks.com

About the Author

Mike Woodward is a technical manager at RadioScape. He holds a B.Sc. in Physics and an M.Sc. in Microwave Solid State Physics and can be reached at mike.woodward@radioscape.com.

