The Use of RF Signal Simulation in a Radio Telescope Control System

Mark H. Clark[a]

National Radio Astronomy Observatory, Green Bank, WV 24944-0002, USA

ABSTRACT

In addition to reliably controlling hardware, a control system should instill confidence by clearly reflecting the user's commands. If the control system of a radio telescope is capable of simulating the effects of the electronics on the RF signal, the user can be provided with practical descriptions of his or her observing configurations. A simulation allows a direct characterization of the RF signal representation rather than a raw list of attenuator, mixer or filter settings. However, because simulation is practical only if it can be kept current and accurate; it must keep pace with both engineering and operational modifications. This is possible if the software interfaces for each telescope device are identical, thus permitting hardware enhancements in the simulation to be implemented as formulated additions rather than as changes. The design of the portable monitor and control system, Ygor, used by the Green Bank Telescope treats each telescope device as an independent unit with identical control interfaces. Differences among devices are reflected by distinct sets of control Parameters. Those Parameter subsets that affect the RF signal representation are passed on to a simulation program which computes basic frequency characteristics throughout the telescope. The signal descriptions are provided to the observers as feedback both in the user interfaces and as part of their data.

Keywords: telescope control, simulation, software, user-interface

1. MOTIVATION

A radio telescope consists of many independent devices built by different engineers for different purposes, sometimes spanning decades, and are used in various configurations often in ways not originally planned. The need to configure such a hodgepodge of electronics, mechanics, interfaces and interacting pieces provides ample opportunity for inconsistent and erroneous sets of specifications. A problem for designers of control software is how to help the user, especially the occasional user, configure the telescope to successfully acquire data.

Typical strategies to assist the observer include providing an expert user to guide the observer, providing a small number of stock configurations which are familiar and guaranteed to be correct, providing immediate feedback in the form of "on-line" data reduction, and observing well-known sources that produce unambiguous and expected results. Of course, these strategies are not mutually exclusive, and most often some combination is employed for optimal use of the telescope.

The latter two strategies listed above require that a test observation be run, and the final strategy even requires the observation of a source that has no interest to the observer. Test runs are important tools, but a more efficient use of the observer's and telescope's time is to employ them as a final check of equipment and ambient conditions rather than as an initial verification of the observer's configuration. A simple strategy, as much as the control system provides the capability, is to allow the user to double check his or her configuration prior to observing. This becomes more effective if the control system can provide more meaningful feedback beyond a parroting of the configuration values specified by the observer or by some observing tool, i.e., the system derives and reports values which more closely reflect the observer's intent.

On a radio telescope, a prime candidate for such inferred information is descriptions of the expected RF signal at any device from the receivers where the RF signal is detected to the `backends` where the RF signal is digitized. Software can provide full availability to a simulated signal all along the signal's path rather than depending on access to physical signals whose attributes are generated from expensive analysis equipment or

[a]E-mail: mclark@nrao.edu

from the actual running of a scan. And unlike a physical signal which can be affected by factors other than the user's specifications, a simulation can provide indicators dependent only on the user's configuration. Specifically, for any radio telescope having significant analog circuitry, the user needs only to specify a configuration for the simulation to check for expected RF signal characteristics. From the control software's perspective, a simulator can provide this information at any point in the system, and configuration verification is not confounded by possible hardware problems. For example, given that the simulator is reliable, if the user found no simulated signal where one was expected, the cause must be a configuration error; on the other hand, if the user saw no physical signal where the simulator indicated one should exist, a problem must exist within the telescope.

A simple simulation driven only by the configuration variables can provide the user with an indicator of the veracity of the configuration prior to running a scan, in fact, prior to activating the variables in hardware since the settings in the control software determine the calculations used in the simulation. As the user configures and modifies the settings across various devices, the simulator can provide continuous feedback on the effect the settings will have on the RF signals.

The availability of RF signal simulation as part of the control system is especially instrumental on the Green Bank Telescope (GBT)* because of the telescope's flexibility, frequency range, and multiple uses. The GBT, by changing the optics and rerouting RF signals, allows the selection of one of nine receivers, each producing two to eight signals. These bandpasses can be partitioned, filtered, and mixed to feed a half dozen data collection systems, each with eight to forty possible inputs. Though having a collection area of 7850 m^2, the telescope is designed for use at frequencies from 100 MHZ to 100 GHz. It is capable of running experiments from continuum studies to spectroscopy and pulsar work and, in conjunction with other telescopes, works as an element for interferometry and as a receiver for radar studies.[1,2]

2. STRATEGY

The building blocks for implementing the RF simulator are based on the fundamental components under user control which modify and route the RF signal in the electronics, i.e., at the level of switches, mixers, or filters. By using such elementary units for its foundation, the simulator's derivation of feedback values is more reliable and more general because of its sole dependency on the base configuration variables which also directly control the electronics. However, because this approach is closely tied to elementary states and events in the control system, it is not helpful in initially translating user's intentions into appropriate configurations.

On the GBT, a wide range of core observing modes has been identified and captured in a configuration program.† This program accepts a specification as a short list of observing values which it interprets and then generates a "best" configuration for the specified observing. The requirements for the program are derived from engineering recommendations and the experience of staff astronomers. Because operation of the program is based on known observing modes, the configuration program is expanded as new observing methods are developed. The program in a very real sense reflects the currently agreed upon "best practices" for the GBT.

On the other hand, the simulator does not contain or rely on observing strategies or even a sane configuration, but it simply reflects at the most elementary level the current configuration. It is blind to observing strategies or optimal configurations since its feedback is built on basic components. Because the simulator is dependent only on the lowest layer of the control software, it is always available for observer and developer alike. Beyond standard observing modes, the feedback from the simulator is useful for developing or enhancing the configuration program, for analyzing problems during observing, for engineering purposes, and for handling non-standard configurations.

The reliance on basic components also infuses an implicit stability into the program. The addition of new circuitry to the simulator does not entail global changes to the program, but only the appending of new circuit descriptions. The nature of appending new circuits is described below in Section 5.2.

*Robert C. Byrd Green Bank Telescope, http://www.gb.nrao.edu/GBT/GBT.shtml

†http://wiki.gb.nrao.edu/bin/view/Knowledge/ConfigurationAPI

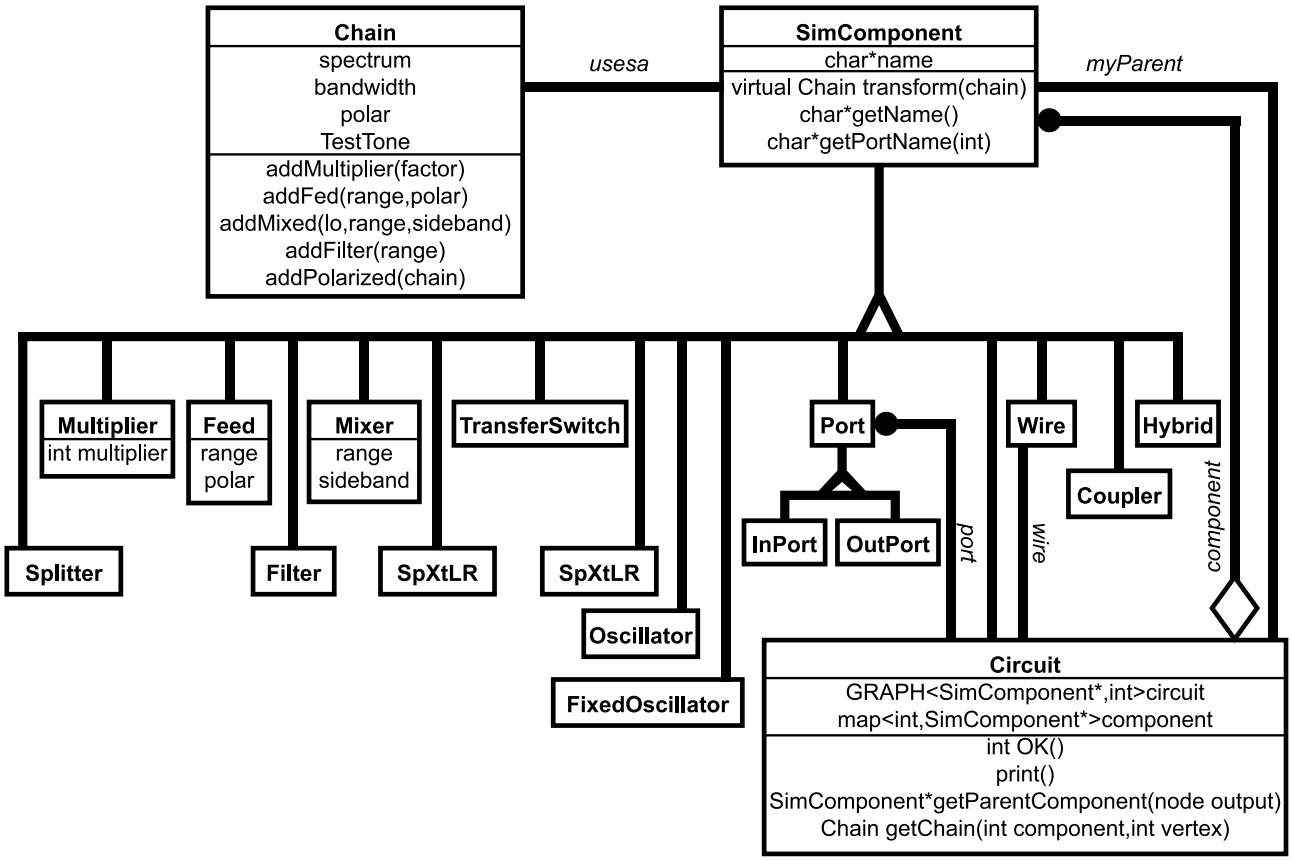

Figure 1. Electronic simulation components object diagram.

3. DESIGN

Simulation components written as C++ classes are used to mirror and to predict the effects of the electronic components in the system. It turns out that, at least for the GBT, only sixteen simulated component types are needed to provide sufficient coverage of the entire RF system. As shown in the object diagram in Fig. 1, each simulated component is derived from the root class `SimComponent` and its virtual class function `transform()`. Each derived class redefines transform() to simulate the component's effect on an RF signal, represented by the class `Chain`. Employing the Composite pattern,[3] any set of simulated components can be combined into a simulated circuit, and sets of simulated circuits may be combined to create new ones until the entire circuitry of the telescope is described by a single hierarchical circuit.

The simulator needs to describe the frequency characteristics of RF signals in two domains: the frequency scale of the RF signal as received at the feed called the `sky frequencies` and as transformed by mixers at any point in the electronics called the `intermediate frequencies`. The RF signal itself can be described by nine variables as listed in Table 1.

This unambitious representation allows the actual computations implementing the simulation to be simple, consisting only of addition and subtraction with conditionals. The RF signal is assumed to be linear so the bandpass can be represented by a rectangular function. For example, feeds define the original bandpass where the sky and intermediate frequencies are identical, filters are assumed to be ideal so they simply truncate the range of the bandpasses, mixers shift and possibly invert the intermediate frequencies' bandpass, and hybrids reverse polarizations. Note that signal amplitude is not simulated because of the additional computational load and minimal usefulness as feedback.

Table 1. RF Signal Description Variables.

"left" limit of the sky frequencies bandpass
"right" limit of the sky frequencies bandpass
"left" limit of the intermediate frequencies bandpass
"right" limit of the intermediate frequencies bandpass
polarization (X, Y, right, left or none)
sideband (whether frequencies run from low to high or high to low)
Boolean indicating the signal's presence
frequency of a test tone in the sky frequencies
frequency of a test tone in the intermediate frequencies

On the GBT the most complex aspect of the RF simulation is routing because of the number of possible paths through its 112 circuits and 364 cables which represent the analog system from reception to digitization. The simulation works by tracing a path upstream from a destination step-by-step through each component to an origin. If the origin is a signal source, i.e., a Feed, it retraces its path downstream accumulating the effect of each component on the signal via the component's transform() class function. Because of this design, not only can the simulator provide the end result – a description of the RF signal as listed in Table 1 – but it can also collect and provide an itinerary of its entire route including each component's name, circuit's name, and the simulation computations applied (see Fig. 6). This list is available at any circuit's ports whether the path carries a signal or not.

An effective final check for insuring the correct operation of the telescope is to inject a single-frequency test tone at the feed and to check for its appearance in the expected channel in the data. Simulation of both the RF signal and any injected test tone is possible since the generation and routing of the test tone are part of the software configuration and therefore available to the simulator.

4. CHALLENGES

If the purpose of an RF simulator is to instill confidence in the user, it must be reliable. Once the program is initially debugged, the primary source of errors in the simulation software is likely to be introduced during modifications required by changes on the telescope. The simulator must be tied closely to the control system, but still be easily adaptable in order to keep pace with telescope modifications, including engineering changes such as new circuitry and operational changes such as cabling changes. In at least one case where a telescope had a simulator, it was abandoned because of the effort required to keep it "current."[‡]

5. IMPLEMENTATION

Primary concerns of the implementation are: Defining of the interface to the control software for obtaining the variables describing the configuration, classifying changes affecting the RF signals and how each type should be handled, and optimizing the speed of the simulation calculations.

5.1. Control Software Interface

A concern for implementing a maintainable simulator is defining a consistent interface to the control software. The interface should be simple, and the inclusion of additional configuration variables should be implemented using a formulated approach. The GBT control system is built using a portable control system, `Ygor`,[4–6] consisting of an aggregate of independent software modules, called `Managers`, which provide identical interfaces for each device on the telescope. Through its Manager, each device is controlled by the same set of methods (e.g., `start()`,

[‡]Jeff Hagen, personal correspondence

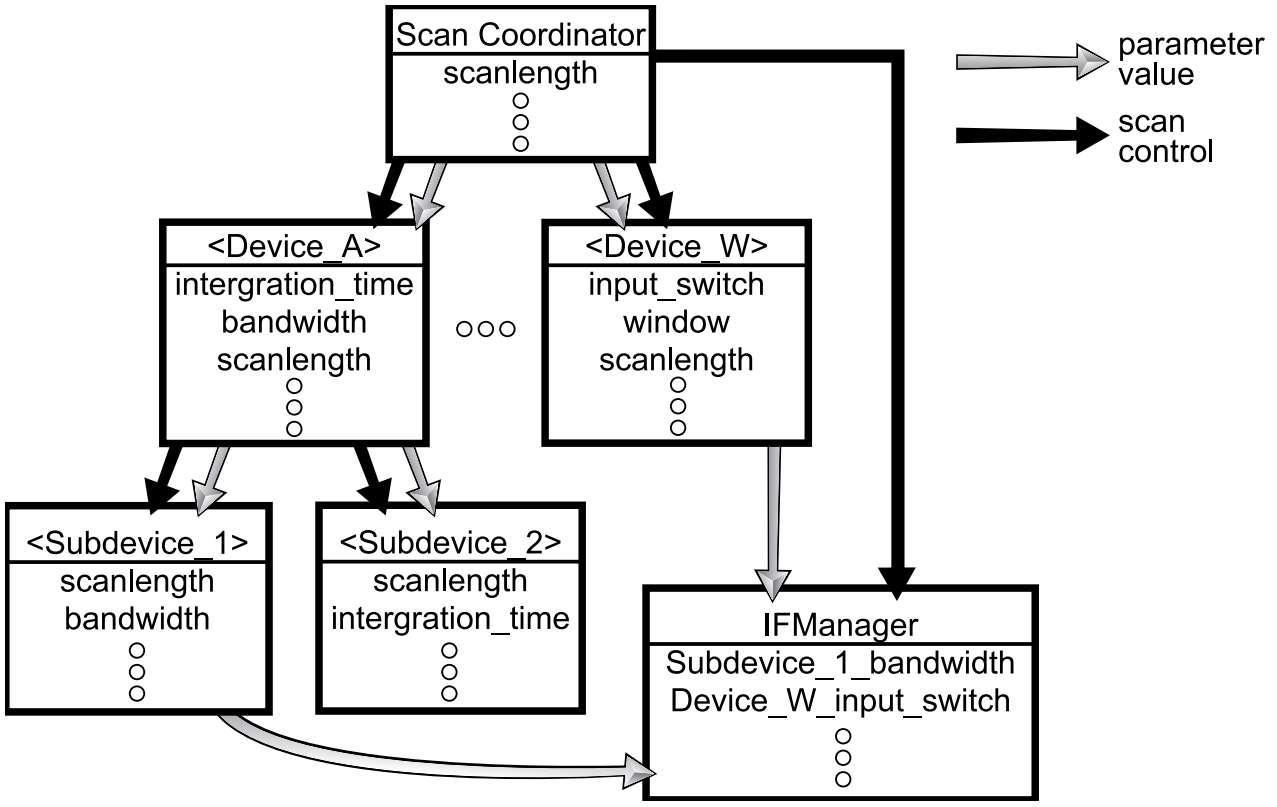

Figure 2. Flow of control and Parameter values.

`stop()`, `setParameter()` or `prepare()`) for initiating all actions needed to perform a scan, but also by a unique set of device-specific configuration variables, called `Parameters`, to characterize the device's behavior during the scan. Though each device may have a different set of Parameters (the antenna, for example, would have a very different set from a receiver or a correlator), the common class function setParameter() for setting these values is identical across all Managers. To any client software of an Ygor's Application Programming Interface (API), one device differs from another only in its set of Parameters.

The Manager's scan methods are recursive allowing Managers to be organized as a tree with scan control flowing from the root to all devices through their associated Managers. In addition, Parameters shared among devices, and therefore common to their Managers, may be set once in the root Manager, called the `ScanCoordinator`, and recursively passed through the tree by a standard mechanism (Fig. 2). For example, the Parameter `scanLength` defines the duration of a scan in seconds and is included by default in the list of Parameters for every Manager making up the telescope. Its value is set only once in the root Manager which passes it along to all of its sub-Managers having that Parameter.

This same mechanism is used to provide the RF signal simulator, encapsulated in the Manager `IFManager`, with all the information needed to reflect those portions of the configuration which affect transformations on the RF signal. Note in Fig. 2 that though the flow of Parameter values does not form a tree because of the need of the IFManager to obtain information from many Managers across the telescope, the flow of control for running scans, however, does maintain the form of a tree with the IFManager being treated as just another Manager.

5.2. Change Taxonomy

A second concern of implementation is how the simulator handles the three types of changes which affect the RF signals in the telescope: configuration changes by the observer, cabling changes by operations, and circuitry

changes by engineering (see Table 2). Configuration changes are handled by the Manager's Parameter passing mechanism described in Section 5.1.

Table 2. Changes affecting RF sigals and their simulation.

Change	Activity	Occurrence	Actual	Simulator
configuration	observing	scan (continuously)	observer initiated	-
cabling	operating	maintenance (weekly)	manually	file editing
circuit	engineering	development (months to years)	construction	programming

Cabling changes, i.e., non-soldered wires between circuits, are listed in a cabling file which is read by the IFManager and must be kept current by whomever is responsible for cabling. Each line in the file represents one cable and is of the form:

`<from_circuit_name>:<port_name> -> <to_circuit_name>:<port_name>`

as seen in Fig.3. All connections between circuits in the GBT's simulator are treated as cables.

```
ConverterModule16:J3 -> SamplerFilter8:J2     # WC16
ConverterModule16:J4 -> NC
ConverterModule16:J5 -> BCPM1:B2              # WC549
ConverterModule16:J6 -> ConverterFilter16:J1  # WC32
ConverterModule16:J7 -> ConverterRack:J16     # WC593
ConverterModule16:J8 -> NC

# ConverterRack AB select switch to SpectralProcessor down converters
# Note: Commented out 1/2/2003 as BPFs inserted -- R. Norrod
#ConverterRack:J17 -> SP_SSBdownconverterA1:IF_input            # WC92
#ConverterRack:J18 -> SP_SSBdownconverterA2:IF_input            # WC93
#ConverterRack:J19 -> SP_SSBdownconverterA3:IF_input            # WC94
#ConverterRack:J20 -> SP_SSBdownconverterA4:IF_input            # WC95

#Declare Spectral Processor External Filters
DEVICE Filter SPbpfA1 205 295
DEVICE Filter SPbpfA2 235 265
```

Figure 3. Cabling file entries. The file format consists of a list of unidirectional cables specified by two circuits' ports; the character `#` denotes the beginning of a comment, e.g., cable names; the string `NC` denotes no connection; and simple components such as filters or splitters can be declared and referenced in the cabling file.

Changes in circuitry, usually in the form of new circuits, are handled by adding new code to the simulator. A circuit, whether implemented as a circuit board, wire-wrap board, or rack module, is represented as a `Circuit` class and its associated components as shown in Fig. 4 and Fig. 5. The actual code is fully contained in a C++ header file and is compiled as part of the simulator. Note that the translation from circuit diagram to a Circuit class is fairly straightforward. The program variable `key` seen in the Circuit class definition in Fig. 5 is an array by which IFManager Parameters are associated with the simulation components they control.

If new electronics are built, new code is written and the simulator is recompiled; if a cable is moved, a text file is updated; and if the observing configuration is changed, the simulator is updated via Ygor's Parameter passing mechanism.

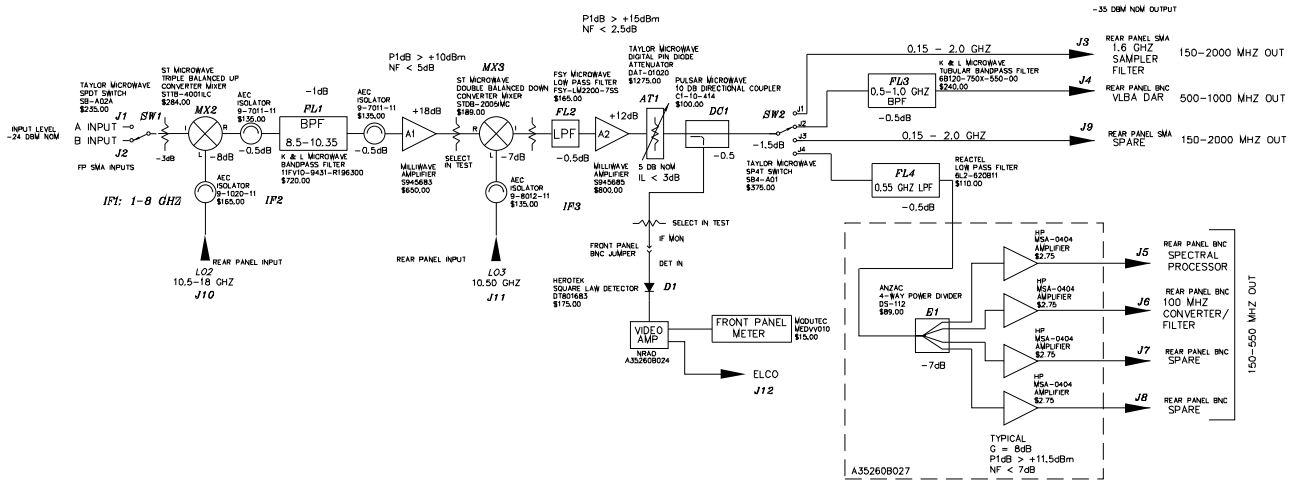

Figure 4. Converter module schematic.

```
class ConverterModuleCircuit : public Circuit
{
public:
      enum
      {
              input = 0,
              output
      };
      ConverterModuleCircuit(int *key, int base, char * name);
};

// The 11 is the number of I/O ports on the circuit.
inline ConverterModuleCircuit::ConverterModuleCircuit(
                                      int *key,
                                      int base,
                                      char * name) : Circuit(name,11)
{
      // components

      // Mixer
      int mx2 = newMixer("MX2",10500.0,18000.0,MixerDef::lower);
      int mx3 = newMixer("MX3",10500.0,10500.0,MixerDef::lower);
      // Filters
      int fl1 = newFilter("FL1",8500.0,10350.0);
      int fl2 = newFilter("FL2",0.0,2200.0);
      int fl3 = newFilter("FL3",500.0,1000.0);
      int fl4 = newFilter("FL4",0.0,550.0);
      // Attenuators
      int at1 = newAttenuator("AT1");

      int swInput = newSpXtLR("SW1",2);
      key[base+input] = swInput;

      int swOutput = newSpXtRL("SW2",4);
      key[base+output] = swOutput;

      // Power divider
      int e1 = newSplitter("E1",4);

      int i = 0;
      // IF input ports
      int j1 = newInPort("J1",i++);
      int j2 = newInPort("J2",i++);
      // IF output ports
      int j3 = newOutPort("J3",i++);
      int j4 = newOutPort("J4",i++);
      int j5 = newOutPort("J5",i++);
      int j6 = newOutPort("J6",i++);
      int j7 = newOutPort("J7",i++);
      int j8 = newOutPort("J8",i++);
      int j9 = newOutPort("J9",i++);

      // LO input ports
      int j10 = newInPort("J10",i++); // LO2
      int j11 = newInPort("J11",i++); // LO3

      // connections
      newPath(j1,0,swInput,1);        // input 1 to switch wiper
      newPath(j2,0,swInput,2);        // input 2 to switch wiper
      newPath(swInput,0,mx2,0);       // output of input to mixer
      newPath(mx2,2,fl1,0);           // mixer output to filter
      newPath(j10,0,mx2,1);           // mixer LO input
      newPath(fl1,1,mx3,0);           // filter 1 to mixer 3
      newPath(j11,0,mx3,1);           // mixer LO input
      newPath(mx3,2,fl2,0);           // mixer to low-pass filter
      newPath(fl2,1,at1,0);           // low-pass to attenuator
      newPath(at1,1,swOutput,0);      // attenuator to output switch
      newPath(swOutput,1,j3,0);       // Output switch to 1.6GHz sampler
      newPath(swOutput,2,fl3,0);      // Output switch to 1.6GHz sampler
      newPath(swOutput,3,j9,0);       // Output switch to 1.6GHz sampler
      newPath(swOutput,4,fl4,0);      // Output switch to filter 4
      newPath(fl3,1,j4,0);            // VLBA DAR filter to J4
      newPath(fl4,1,e1,0);            // filter 4 to splitter
      newPath(e1,1,j5,0);             // Splitter to output ports
      newPath(e1,2,j6,0);
      newPath(e1,3,j7,0);
      newPath(e1,4,j8,0);
}
```

Figure 5. Circuit class for a converter module.

5.3. Optimization

The last major concern of implementation is simulation optimization. For feedback to be useful, it must be an immediate reflection of the user's actions. Speed of individual component simulations is important because of the number of simulations needed for the telescope. In the GBT, over 1500 signal descriptions are made available for every detected change. The simulation computations are faster because of the simplicity of the computations used to generate signal descriptions; both the signal representations and therefore the required computations are kept to the bare essentials.

As described in Section 3, the simulator computes the RF signal representation by accumulating the effects on the signal for each leg of the path that the signal traverses. Since the signal path branches out as it travels from feed toward the backends, many of the resultant RF signal simulations share common circuitry and, therefore, common intermediate simulation results. In fact, in the process of computing the RF simulation for a specific port, the simulator has to compute the RF simulation for every port along the path between it and its origin. The simulator takes advantage of this situation by internally saving the accumulated results from each circuit's output port, thus removing the need to repeat the computations up to that point. On the GBT, though over 1500 RF simulations are made available, no more than 300 simulations (the total number of circuits' output ports) are computed.

6. USES

6.1. Feedback

The primary purpose of the GBT RF simulator is to provide meaningful feedback to the user. An example application of such feedback can be seen in Fig. 6 which shows a dialog generated by the GBT graphical user-interface program CLEO.[7] This dialog is available at each displayed circuit's input and output port.

6.2. Data

On the GBT, the information generated by the simulation is also used in data reduction. Because the GBT allows the observer to control the electronics to the level of individual components, the total effect of these components is needed to determine the characteristics of the resulting RF signal. The equation representing the transformation between the initial sky frequencies and the final intermediate frequencies at the backend is called the `Sky Frequency Formula`. The coefficients for the equation are directly derived from the sometimes lengthy and convoluted chain of electronics the signal has transversed. For every complete RF signal path, i.e., for every full path from a receiver's feed to an input port on a backend, the IFManager generates not only the coefficients for the Sky Frequency Formula, but all the information the simulator has accumulated for that path and writes it to a data file for use in data reduction and, if problems occur, postmortem analysis.

6.3. Warnings

Besides generating RF signal descriptions, the simulation is able to detect and generate warnings when a configuration contains discrepancies. By the backends passing on to the IFManager a Parameter listing their active input ports, the simulator can determine whether the configuration provides viable signal paths from feeds to backends. If the simulation of an active backend port fails to produce a signal, an error is immediately generated. Likewise, by checking the bandwidth of the simulated RF signal, a warning is generated when the system is operating outside the nominal bandpass. Only a warning is generated because such configurations can be used by observers to investigate frequencies on the edge of the telescope's bandpass capabilities.

7. CONCLUSION

The primary intent of the simulation system was to provide useful feedback to the user during configuration. But the universal ability of the simulation system to generate correct Sky Frequency Formula coefficients reliably, even for new configurations, turned out to be important in pilot observing during commissioning. The Ygor control system was designed prior to any consideration for writing a built-in RF analog simulation program, but the design goals for the control system, namely, a coalition of fully independent parts with identical control interfaces, made introduction of the simulation program as part of the control system straightforward. A drawback in the

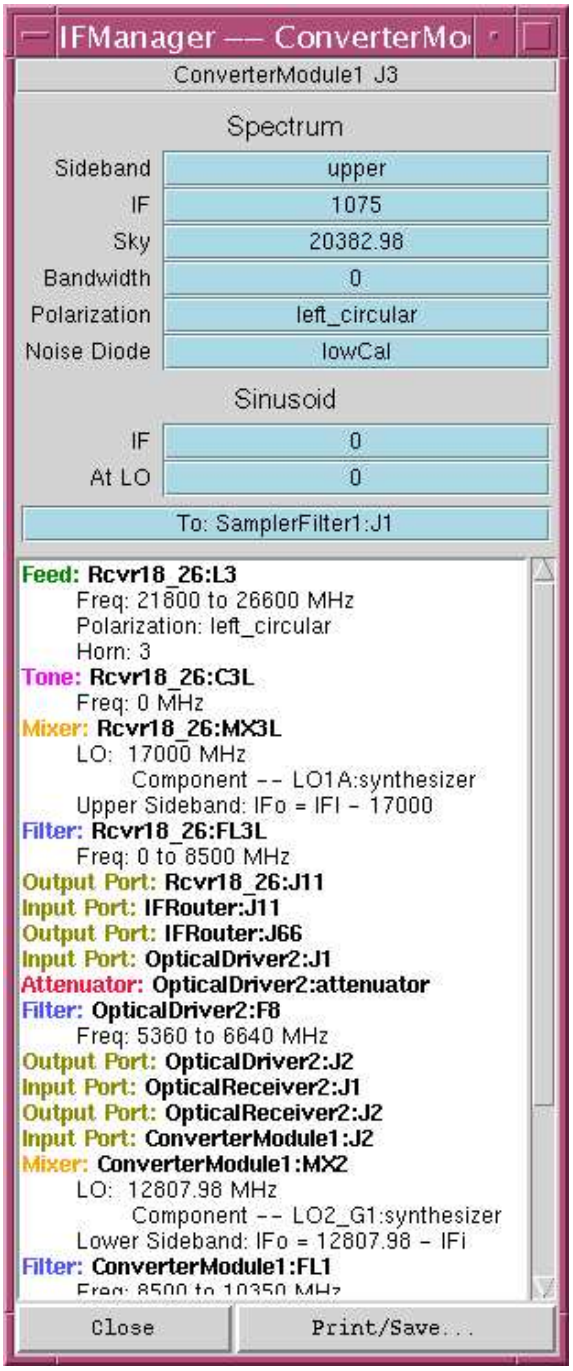

Figure 6. CLEO dialog displaying an RF signal simulation description. The signal described is at output port J3 of ConverterModule1. The mixers between feed L3 and port J3 have shifted the center frequency from 20383 MHz to 1075 MHz. The bandwidth is 0 MHZ because the signal is outside the nominal limits of the electronics. The signal at the junction displayed has traversed four circuits: Rcvr18_26, IFRouter, OpticalDriver2, and ConverterModule1.

design is that errors introduced when adding new circuits are conspicuous, but finding the location of the problem is often laborious. The current user-interfaces to the IFManager are based on the user querying for information from the simulator at specific points on the telescope as shown in the dialog in Fig. 6. More could be done to enrich the user-interface displays with inherent RF information. For example, lines representing possible RF signal paths could indicate whether a signal is present or could display simple characteristics such as center frequency or bandwidth. Though the effort to augment the simulator with new circuits pales against the effort for actually designing and constructing these circuits, the simulator could be made more robust if the descriptions of the circuitry were implemented as data rather than as code.

ACKNOWLEDGMENTS

Most of the requirements for the early work on this program were extracted from an unpublished GBT memorandum by Carl Heiles and Ronald J. Maddalena. David Rose led the effort by GBT operators in initially translating many of the GBT circuits into the circuit descriptions needed by the simulator. J. R. Fisher and D. Anish Roshi helped me interpret some esoteric circuits so that they could be handled by the simulator's set of components. Eric Sessoms helped refactor most of the simulation code and a much improved object-oriented design resulted. Finally, many hours of testing by NRAO staff astronomers verified the accuracy of the simulation.

REFERENCES

1. P. R. Jewell, "The Green Bank Telescope," *Proceedings of SPIE.* **4015**, p. 136, 2000.
2. F. J. Lockman, "The Green Bank Telescope: an Overview," *Proceedings of SPIE.* **3357**, p. 656, 1998.
3. E. Gamma, R. Helm, R. Johnson, , and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, 1995.
4. M. H. Clark, "The Control Software Architecture for the Green Bank Telescope," *Proceedings of SPIE.* **3351**, p. 287, 1998.
5. M. H. Clark, "A Radical Approach to Software Modularity in Telescope Control Software," *Proceedings of ADASS VIII.* **172**, p. 87, 1999.
6. J. R. Fisher, "Object-Oriented Experiences with GBT Monitor and Control," *Astronomical Data Analysis Software and Systems VII, A.S.P. Conf. Ser.* **145**, 1997.
7. R. J. Maddalena, "The User Interfaces for the NRAO-Green Bank Telescope.," *Proceedings of SPIE.* **4848**, p. 316, 2002.