The FSM Virtual Radio Kernel:

Why, What, & How

(In That Order)

Frank Brickle, PhD, AB2KT

DTTS Microwave Society

6 Kathleen Place

Bridgewater, NJ 08807

ab2kt@arrl.net

July 2007

Abstract

Software Defined Radio is moving fast...in some areas. In many important ways, however, Software Defined and Cognitive Radio are just now arriving at the "horseless carriage" stage, where software is being embraced mostly as a *substitute* for traditional circuitry, or is being celebrated largely for what are, after all, only *incremental enhancements* to a very conventional user interface. Software Defined and Cognitive Radio promise more than this. We outline the technological and conceptual components that are now making it possible to take the next steps towards those promises, and we report on current ongoing work at forging these pieces into a new kind of radio environment.

Prologue

Let me begin by quoting a thoughtful note from Roger Rehr, W3SZ. In the note, Roger is speculating on what his ideal VHF-and-up contest station would look like from an operational standpoint. There are many reasons to take Roger's speculations seriously. Most important here are the facts that, first, Roger is a very successful contester; second, he's one of the earliest adopters of SDR in amateur radio; and third, he's been one of the most persistent and generous *users* of the bleeding-edge and often faulty tools that we developers have been putting out. I've edited the text slightly to emphasize how the requirements aren't really bound to a single kind of radio or software platform.

> *My application (as you know) will be to have several SDRs, so that I can constantly see bandscopes on 50, 144, 222, 432 MHz regardless of whether or not I am on one of these bands (or on the microwaves), and so I can instantly switch to one of these bands if a new signal pops up. The station itself operates on all bands from 50 MHz thru 24 GHz with automatic bandswitching controlled by the logging program.*
>
> *The features of the software console that I think would be needed for this application include, listed in decreasing order of importance, the following:*

(1) Ability to simultaneously display several instances of the console each running its own SDR front end all on a single computer

(2) Ability to see panadapter displays of both the Main Receiver and the Optional Second Receiver simultaneously (so 2 bands can be watched with one SDR front end).

(3) Ability to take transmit audio and PTT signals from computer and to direct them to whichever instance of the console 'has the window focus' so that one mic and PTT switch, connected to the computer, can be used for all instances of the console. Perhaps 'following the focus' would be a checkable function option, as there might be times that one did NOT want to follow the focus in this fashion e.g. if doing talkback on 144 MHz while searching for a signal on the 10 GHz bandscope.

(4) *Ability to easily mute/unmute audio from each instance of the console when switching between instances (or even better have control of position of each instance in the stereo soundscape). This should be simple but manual, as I might want to be monitoring a 6 meter station (for example) while running on 2 meters. If this followed the focus automatically like PTT and mic one wouldn't be able to do this.1*

All in all, a pretty intricate scenario. Those of you who already SDR users will recognize what an enormous leap such an operating environment represents from the “glass front panels” we currently have to rely on. For example, consider Illustration 1, the PowerSDR Console[2] used with the SDR-1000, the FLEX-5000, HPSDR, SoftRocks, and other things -- and it's truly a very good example. But look at all those buttons! And this interface can't even be resized. And it only runs on Windows, at the moment.

Fear and trembling. The idea of designing, writing, debugging, documenting, and shrink-wrapping a completely flexible version of an application of such complexity is truly intimidating – until, that is, you recognize that, unlike the PowerSDR Console, Roger's scenario isn't a single application at all; it's a constellation of smaller specialized applications. Furthermore a large part of the scenario – the critical infrastructure to make it all work together – already exists in a mature, hardened form. And a good part of what's left exists and, while it is still somewhat experimental, it has already made its way out into userland and is being subjected to the kind of hammering that will make it smooth and trustworthy before long.

Our subject here, then, is the infrastructure for making Roger's station a reality. Well, no. Our subject is actually the conceptual framework needed to shape and leverage the infrastructure and make it usable for our purposes. We're going to call such a framework a *Virtual Radio* or *VR,* and an SDR that's realized in it will be a *VR-SDR.* Our particular version is called *FSM*, for reasons that will become clear eventually.[3] Our main focus will be the central organizing piece of the VR, the *VR-Kernel*. We call it that because it has the same responsibilities vis-a-vis the rest of the VR, as the kernel of an underlying operating system has to the rest of the computer. In particular, a VR-Kernel will be the brain center for potentially many VR-SDRs and many users, very probably coexisting on a broadband network.[4]

1 Personal communication from Roger Rehr, W3SZ on May 26, 2007.

2 http://www.flex-radio.com

3 It's not a coincidence that our kind of VR has the same shorthand name as Virtual Reality.

4 We have an acronym for the entire VR-SDR system, *GBG*, but the explanation for that will have to remain mysterious for awhile.

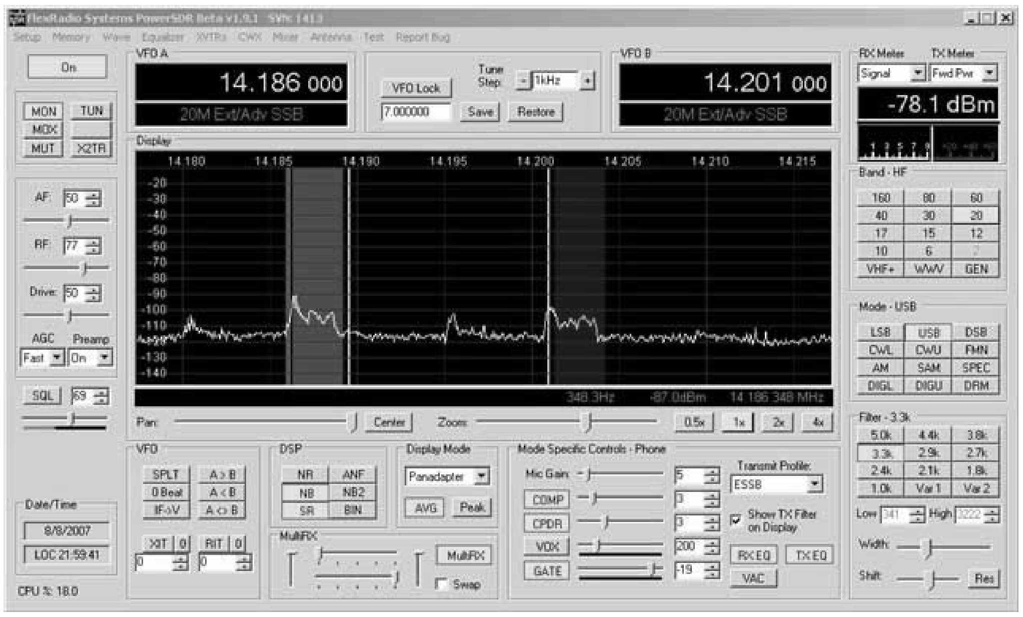


Illustration 1: The PowerSDR Console.

Digression Concerning Precedents

You may ask yourself, “How can he be so sure about this?” The answer is simple: we've been through this before. A generation ago, digital signal processing for musical purposes was moving out of the laboratories and into the hands of working musicians. The same kind of relationship obtained among the developers, the applications, and the users of music software, as holds within the SDR community today. To follow the musical tale through to the present would be an interesting but labyrinthine digression.[5] Therefore let me just assert that, in many senses, the software design and user interface issues faced by computer musicians a generation ago, and which we software radio people are facing now, are “solved.” The lessons are there to be learned and the solutions imitated. For an especially elegant solution set, you can look up PLOrk (the Princeton Laptop Orchestra)[6] and its implementation language, ChucK[7].

One particularly intriguing feature of that system is the capability to create new processing algorithms on the fly *as part of a real-time performance* involving many networked laptops, each being operated by a different musician, and with sample-synchronous coordination of all the performers. This is precisely the sort of capability that genuine Cognitive Radio requires, and it's a relief to know it's not speculative moonshine, and it's usable in a high-pressure situation (a public concert) comparable what one might face in, say, a high-performance contest station.

So What Are We Talking About, Then?

A lot of work is involved in making the transition from glass-front-panel SDRs to VR-SDRs. Overall this work falls into three major areas:

5 One of the tangential lessons is that, as far as signal processing is concerned, the most efficient optimization technique is to wait for the hardware to get faster. You won't be waiting long.
6 http://plork.cs.princeton.edu
7 http://chuck.cs.princeton.edu

1) Refactoring SDR applications,
2) Building the operating environment in Erlang/OTP,
3) Exploiting 3D graphics support.

In the following sections we will consider each of these in turn.

Refactoring SDR Applications

***Where we are now*.** What you have in a typical SDR transceiver today are three major logical components: the RF front end, the DSP core, and the User Interface/Control mechanism. This skeletal structure is depicted in Fig. 1. As this diagram shows, the "brains" of the SDR-as-radio are centralized in the UI/Control component. This is where the behavior of the radio, if you will, is implemented. Part of what the UI/Control component does is translate between emulations of traditional radio controls (buttons, meters, knobs, etc.) and the widely disparate vocabularies and sentences that the DSP Radio Core, AF Hardware, and RF Hardware understand. It's important to keep in mind that this control is still all digital and in software, although it's not signal processing code *per se*. The only analog path in this diagram is the one between the AF and RF components.

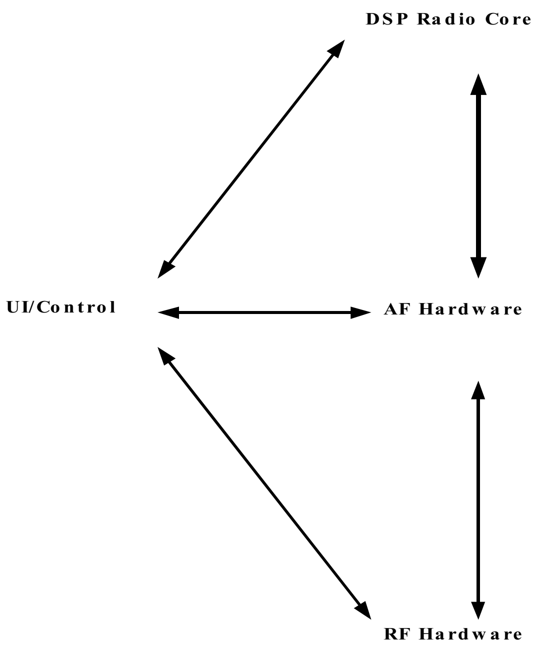


Fig. 1. General Outline of a Conventional SDR

One of the conspicuous features of this arrangement from a software-design point of view is that it's all *tightly coupled.* To put it crudely, about the only thing standing between any one software component in this diagram and any other component is an API. There are *ad hoc* situations where a connection can be faked – that is how virtualization and remoting are done in this scenario – but, all in all, the design is basically that of a monolithic application occupying a flat address space on a single machine. Embedded in the picture are myriad unarticulated presumptions what the sinks and sources of each of the components will be.

What makes this picture undesirable is simply that every variation, every customization of the design needs to be explicitly programmed into the application.

***Where we need to be*.** The first thing that needs to happen is for this diagram and its design assumptions to be blown up. Exploded. The pieces need to be entirely decoupled. The capability needs to be established for arbitrarily many of each of the components to be created, controlled, and destroyed on the fly. Even more, the capability of creating and re-creating arbitrary nested *combinations* of these components needs to be a standard part of the repertoire.

If you've ever taken an interest in the abstract theory of programming languages your ears will perk up at these requirements immediately. What these requirements imply, among other things, is the stipulation that each of the small individual components must be *referentially opaque* and *lexically scoped*. The cash-value translation of that would be: each of the small individual components must be capable of running as an independent standalone process. Each little component is its own little independent program. That includes the UI and also each of its constituent subwidgets. While it seems like this threatens to become yet another thicket of complexity, the requirements of referential opacity and lexical scoping are precisely the therapies needed for thinning out complications. The major theoretical gain is that the rules for combining components can be defined recursively[8]. As a consequence, components at any level can be replugged (and often hot-swapped) without interference to the rest of the system.

Under this model, there are zero reasons for the components all to be written in the same programming language. In fact, if we do our jobs right, there are zero reasons for the components all to be running on the same machine. This holds true all the way down to the cold metal of each component.

You want it, we've got it. Go back to Roger's Dream Station[9] for a moment. His requirements (1) multiple radio consoles running on a single computer and display, and (2) multiple panadapter displays for each radio, are considerably simplified if the constituent pieces – the consoles with all of their subwidgets, multiple panadapters attached to each console – can be realized as separate standalone programs. For one thing, it makes development and testing of the pieces much easier. But more important, it means that the constituent pieces can be created, used, and destroyed *at the user level, not at the program level*. It can be accomplished either directly by the user, or by some proxy acting for the user, which would *itself* be an independent, standalone program. That's the VR-Kernel. Of course, management of the pieces to make them act in consort all happens outside the pieces themselves, and that, also, is the VR-kernel. What it actually consists of is the topic of the next section.

A few final comments about this approach.

Already ahead of the game. A lot of the work outlined in this part of the discussion is already done, although very few members of our amateur SDR community are aware of it, and even fewer are concerned with it yet. *DttSP*[10] is the name of the project, led by Bob McGwier, N4HY and myself, that provides a portable, efficient DSP core for general SDR receivers, transmitters, and transceivers. It provides the signal processing capability in PowerSDR, the Open Source SDR application used with the FlexRadio SDR-1000 and FLEX-5000 radios, among others. DttSP was designed from the ground up with decoupling and decomposition into smaller functional units in mind. Controlling the SDR processing is done entirely from a separate user process. Additional pre- and post-processing is patched into the signal chain externally via the JACK audio subsystem. The intrepid experimenters using sdr-core from DttSP on Linux, Mac OSX, and FreeBSD seem uniformly to agree that all of the obstacles this configuration presents are one-time setup issues. The performance impact, if any, is negligible. Multiple instances of it can run simultaneously without problems. The DSP part of VR-SDR is ready to roll.

Pay no attention to the man behind the curtain. There are quite a few components in VR-SDR that a user will never see. These are the pieces that realize the "cognitive" part of a Cognitive Radio. As with other components, the functions that these pieces provide can be constructed out of an extensive array o existing software such as LNKnet[11], autoclass[12], screamer[13], gsl[14], apophenia[15], ghmm[16], lush[17], and

8 Similarly, programs in any modern language can be written to conform to the same constraints.
9 Roger's Dream Station = RDS = SDR backwards? What a coincidence!
10 See http://www.dttsp.org and http://groups.yahoo.com/group/dttsp-linux.
11 http://www.ll.mit.edu/IST/lnknet
12 http://ic.arc.nasa.gov/ic/projects/bayes-group/autoclass
13 http://www.cis.upenn.edu/~screamer-tools/screamer-intro.html
14 http://www.gnu.org/software/gsl
15 http://apophenia.sourceforge.net
16 http://ghmm.sourceforge.net

many other libraries and programs. From just this brief list it should be evident how wrong-headed it would be to insist that such a body of complex and sophisticated software be rewritten in order to fold it into a single application. Far better to let the programs be programs, and to provide a smooth way to integrate them and let them cooperate in the VR-SDR processing flow as they currently stand.

Learning by Doing. A VR-SDR is a complex and dynamic organism. One of the few features we can be certain of, at an abstract level, is that it will be impossible to represent one as a planar graph[18]. This is a kind way of saying that SCA[19] has it totally backwards: you can't define the structure and protocol for a VR-SDR first, and then proceed to program the parts according to the pre-established spec. Furthermore there is no way to effectively organize a total development effort for such an evolving, dynamic creature The appropriate development methodology, despite received wisdom, is bottom-up[20]. As with the Cognitive software components, a way needs to be provided for individual participants to do what they do best, using the tools they prefer, to contribute functioning components to the VR-SDR codebase, without demanding that they upend their effective working methodologies, and without demanding that they grasp all the intricacies of the environment they're working in. In short, single contributors should have to understand only the *constraints* on their specific contributions, not the nested dependencies of the whole VR-SDR system. This is extremely important in enabling the work of individual participants, and their contributions are critical, because, as has often been remarked, “great art is not written by committee.” On purely practical grounds we need to smooth the way for solitary contributors. And so, once again, we have to conclude that a VR-SDR resembles an operating system more than an application, and in this case, we're following the contours of OS design originally established by Unix.

It is absolutely essential that the VR-SDR codebase be Free and Open Source. This is a non-negotiabl requirement.

Erlang/OTP as the Operating Environment

You know what they say about pictures, so here's one that might help. Fig. 2 is a diagram of a VR-SDR in one of its many possible transient configurations.

Not a classless society. In this diagram there are two basic kinds of objects and two kinds of connections. The objects in black circles are functional radio pieces. As far as the VR-SDR is concerned we will call these *nodes*. The grey elliptical thing is the VR-Kernel. It also is a node, but it has a privileged position. The greatest privilege the VR-Kernel has is this: it alone can create, control, and destroy other nodes.

Part of the 'control' privilege is telling newly-created nodes about other nodes, the ones they will be exchanging data with. That's why there are two kinds of connections: one type for control, one type for data. Here then is a basic principle:

> *All* ***control*** *passes through the VR-Kernel;* ***data*** *can pass among other nodes, but it doesn't have to.*

17 http://lush.sourceforge.net
18 There is ample reason to believe a VR-SDR will be best represented abstractly as an incomplete directed graph embedded in a torus.
19 www.spectrumsignal.com/publications/Extending_SCA_Inside_SDR_Modem_Architecture.pdf
20 http://www.paulgraham.com/progbot.html

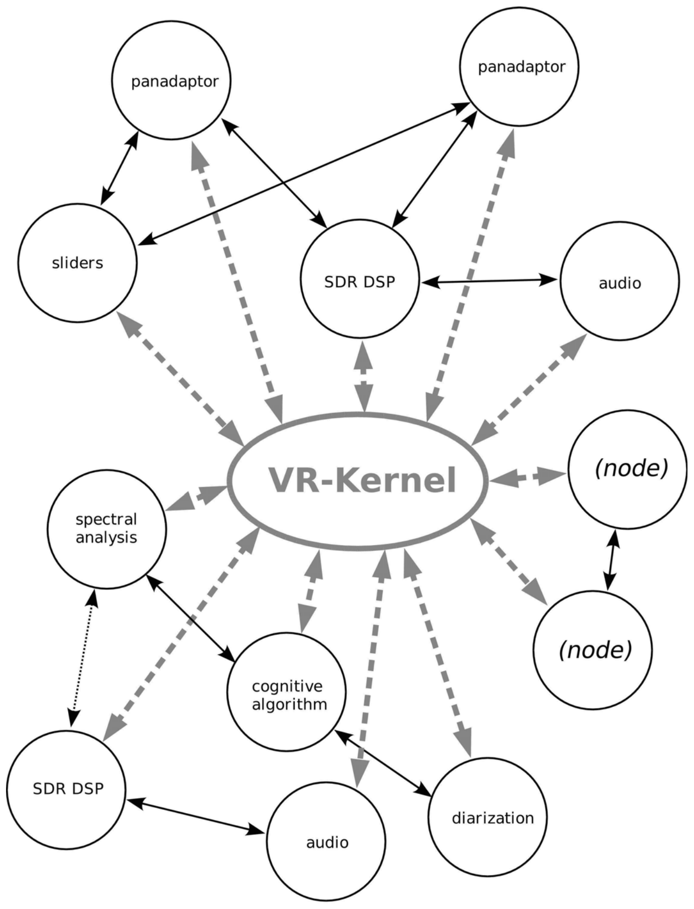


Fig. 2. A typical VR-SDR configuration.

The principle is illustrated in Fig. 2 with two different styles of arrows[21].

There is another important principle that's harder to illustrate, but which justifies our apparent violation of Occam's Razor:

All knowledge of ***current*** *and* ***possible radio states*** *resides in the VR-Kernel.*

In other words, such things as the repertoire of available node types (VR-SDR components), types of data, types of connections, platform hardware -- essentially everything about the entire VR-SDR except the functional algorithms is embodied in the VR-Kernel. Significantly, this includes registering the values of all parameters controlling the nodes, validation limits for those parameters, and so on[22]. A corollary is that the VR-Kernel is responsible for monitoring the system's health, trapping and correcting errors, ensuring graceful system startup and shutdown, and maintaining persistency of system state when it's not operating[23].

21 The criscrossing arrows show why the VR-SDR graph needs to be embedded in a torus.
22 In concrete terms, this means things like “VFO” frequencies, filter band edges, compression levels, etc.
23 That is: there are no free variables in the nodes, and only the kernel can supply values for the bound variables at the top level of any other node. Data sources and sinks are (random) variables.

Ontological Relativity. Notice how, perhaps surprisingly, we're saying nothing at all about how the nodes and connections are actually realized. That's because they don't all have to be the same in those respects. As far as the nodes are concerned, this simply reflects the idea stated before that individual functional components should be programmable according to the taste and predilections of the programmer. The same idea extends here to the nature of the control and data connections, except here, we're taking an agnostic line about where the nodes are running, and thus how they hook up. That is, we fully expect that nodes can and will be running on any and every processor, WAN, LAN, or local, available to the VR-SDR. As a consequence, the same two logical components may be connected by two very different kinds of 'wires' on different occasions. And that's fine with us. What's amusing about the situation is that even the VR-Kernel itself doesn't *necessarily* need to know where a given node – even itself! -- is running at a given time, either, although we're prohibiting that by convention.

Is this for real? So, you may be asking yourself, how do we bring about these wonders? If I tell you, "by writing a few lines of code, but only a few," would you believe it? Maybe not, but you should. The hard part, the bulk of the work is done already. It's provided in grand style by *Erlang/OTP*, the Erlang language and the Open Telecommunications Platform originally developed by Ericsson.[24]

Erlang is a Functional Programming Language[25] with strong roots in Prolog, but with concurrency[26] features from Communicating Sequential Processes[27]. OTP is a complete operating environment, written in Erlang, expressly designed to support robust, efficient distributed telecommunications processing systems. For our purposes, nearly everything in the way of infrastructure necessary for VR-SDR exists in mature form in Erlang/OTP. One especially useful feature is the ability to write Erlang nodes in virtually any modern programming language, without giving up the behaviors that make pure Erlang nodes especially adaptable to large, mission-critical systems. Erlang/OTP is being actively developed and maintained, and it's issued under the GPL. There are satisfactory, interoperating versions for all significant operating systems and architectures.

What have you done for me lately? Rather than lingering on an exposition of Erlang/OTP, let's move ahead with a discussion of where the VR-Kernel is at right now, and how, thanks to Erlang/OTP, it's growing and developing into a useful environment for VR-SDR. The fact that we can "grow" the system underlines a third critical principle:

> *A VR-SDR and the VR-Kernel can be developed and evolve on the fly without restarting the system. Multiple versions of all functions can coexist withing the same running system.*

Admittedly, this has the flavor of Otto Neurath's apothegm[28] about rebuilding your boat while you're floating in it, but it's a crucial property of a complex, real-world system. The world would be a better place if more mission-critical systems adhered to this design constraint.

Rather than invent the world from scratch, what we've chosen to do is start by modeling the behavior of various legacy radios like the Elecraft K2 and the Kenwood TS-2000. We've adopted as the minimal

24 http://www.erlang.org
25 http://en.wikipedia.org/wiki/Functional_programming
26 http://en.wikipedia.org/wiki/Concurrency_%28computer_science%29
27 http://www.usingcsp.com
28 Marie Neurath and Robert Cohen, with Carolyn R. Fawcett, eds. *Philosophical Papers*, 1913–1946.

starting point the CAT command vocabularies of these radios, much as Bob Tracy K5KDN and Bill Tracey KD5TFD have done for the PowerSDR Console under Windows. As they've done, we have expanded the CAT vocabulary to express controls not meaningful for the model radios. Where we start to diverge is when we begin to address capabilities beyond the "single physical radio" model – when we need to control arbitrarily many simultaneous receivers, for example, or when we begin to fold in closed-loop feedback control of the radio operation based on cognitive algorithms. To give one instance, one of the simpler pieces of cognitive radio programming is to allow secondary software receivers which are capable of detecting select kinds of spectral energy. When an event occurs, a receiver can clone itself and direct its duplicate to track the signal and subject it to further processing such as recording to a disk file, demodulate and decode as PSK31, and so on. This is hard to represent using CAT codes, but the basic implementation can still use the CAT vocabulary to express the individual sub-receiver control.

For our next number... In this scenario the VR-Kernel serves mostly as a sophisticated switch for control information. It does have the immediate advantage of being transparently distributable, however, so that CAT codes can originate from or be directed to anyplace on a network. As the repertoire of enhancements grows, it inevitably will reach a point where a new conceptual framework for the radio controls will emerge. One of the first big steps in that direction is developing an alternate, richer command language. Since the FLEX-5000 uses MIDI[29] as its internal command representation, a polyglot language, capable of being transformed into either CAT or MIDI codes, is practically essential. This requirement and more is satisfied by an intermediate representation in XML. (Naturally, Erlang/OTP already possesses facilities for parsing and executing XML strings.) Within the design we'v described, this can coalesce naturally alongside the existing control models, and they all can continue to coexist and while leaving legacy software to die in peace and of old age.

Repertoires like this of emulated behaviors are sometimes called "personalities." One VR-SDR personality emulating the K2/TS-2000 has been in existence for a couple of years, with coded primarily in Python. This personality is actively being rolled over into Erlang at this time with the SDR-1000 as the target hardware, and, with the enhanced command language we described, migration to the FLEX-5000 is a straightforward move.

Perhaps you now begin to get a hint of how Roger's Dream Station can be realized as a dynamic operating environment rather than a single, hopelessly baroque application. Creation, destruction, routing, parameter variation, and focus of control are all under the management of the VR-Kernel. Each of the sub-components functions in its own natural way, but participates simply in the VR-SDR as an Erlang node. Software components can be distributed and re-distributed among processors transparently as the transient operating configuration may require. Given a sufficiently fast network, the entire system can be run remotely, and it can be trusted to perform properly at a remote site since it can monitor its own performance and it can be reconfigured, repaired, and upgraded on the fly, using capabilities built in to OTP.

We are not alone in the universe. It would be very misleading to suggest that our version of VR-SDR was the only one, or even necessarily the best one. The kernel-and-node design we've outlined here is meant to map well into an operating environment such as Roger described. There are other interesting possibilities as well. For example, Bob Cowdery, G3UKB[30] has gone a long way in the development of

29 http://www.midi.org
30 http://myweb.tiscali.co.uk/g3ukb

VR-SDR based on DttSP and Erlang/OTP but with a very different orientation. Bob's goal is to provide comfortable environment for breadboarding multiple parallel SDR designs using fundamental building blocks in different combinations. This design is obviously aimed at experimentation and not contest operation. Neither his design nor the one presented here precludes the other's application domain, but the differences do entail some fairly deep consequences at the level of infrastructure. Bob's design relies on the explicit programming of a Finite State Machine that governs the interaction of components. Our design, by contrast, makes use of the FSM "behavior" that's a major component of OTP. We find this to be no obstacle under the constraint mentioned before, that all control passes through the VR-Kernel, while data can either pass through the VR-Kernel or directly between individual nodes.

A kernel of truth. Speaking of Finite State Machines, it's time to unveil the reasons we call our VR-Kernel FSM. For one thing, it's because the OTP FSM behavior (along with the server behavior) are at

the heart of our design. The real reason, however, is that this cult symbol:
bears an uncanny resemblance to the diagrammatic representation of our kernel. Here FSM is also an acronym for "Flying Spaghetti Monster," which no doubt is how we will be thinking of the project during difficult stretches in the continuing development process.

3D Graphics for the Display and User Interface

One of the giants of musical signal processing, James A. Moorer, has often said that he didn't see how you could do computer music without a lot of graphics. What he does not mean, of course, is that music suddenly becomes a visual art. It's still meant to be heard and read, and reading something is very different from looking at it. What he does mean is that traditional music making has an enormous tactile component which is missing when you move to the computer. One way of making up for the loss is with rich visual information. But that's only partial compensation. Music-making means exercising fine control over many parameters at the same time, no matter what instrument. A generation of history with electronic instruments has convinced everyone that you can't do it all with a bitmapped display and a mouse.

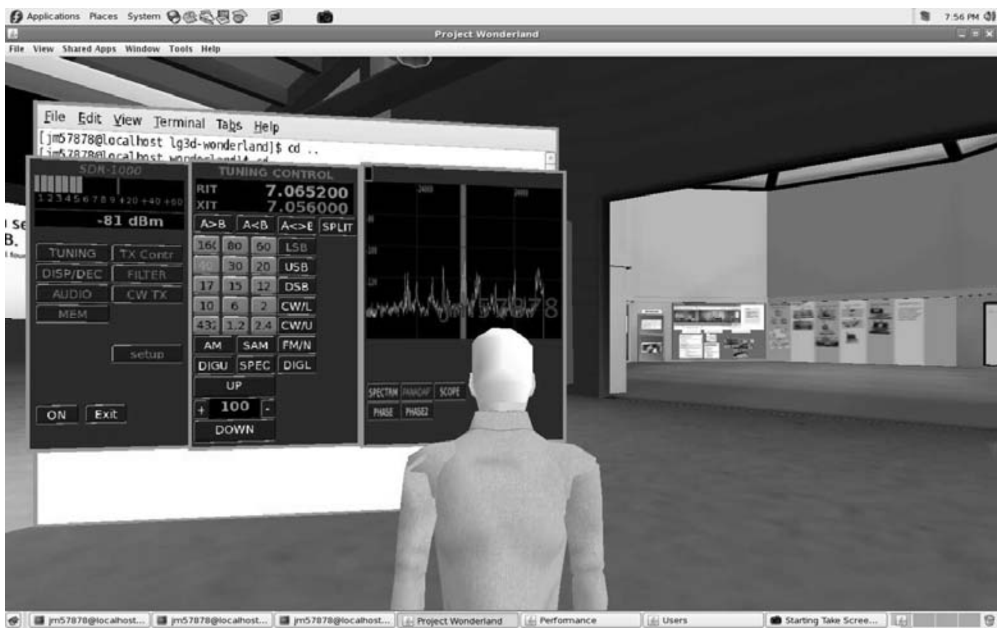
Mutatis mutandis. I would submit that very similar observations could be made about using a radio, especially in competition, and most especially with Roger's Dream Station. For the time being we'll leave complex tactile control out of the picture. What we're addressing here are merely the enhanced visual capabilities which Roger's Dream Station will have to have, both for displaying realtime information and for expressing realtime control. There are five essential capabilities:

1. Representing information and control objects in three dimensions,
2. Reconfiguring all visible objects continuously and in realtime under either user or program (independent) control,
3. Transforming visual objects in realtime,
4. Conceptualizing and representing the entire system of visible objects in a unified three-dimensional space,
5. Having mobility in time also – basically, being able to move around at will in the history of the system.

This should sound pretty familiar. What we're saying is that visually, Roger's Dream Station is at least as complex and capable as a high-performance video game.

Proof of (Second) Life. John Melton, G0ORX/N6LYT, who has been an irreplaceable participant in many if not all of the most significant technical advances in amateur radio for the last decade at least, has darted ahead of the crowd again. What John has done is to give an irrefutable demonstration of how the glass front panel needs to be lifted into three dimensions. His jsdr, written in Java, was one of the first alternative consoles to PowerSDR for the SDR-1000 and the SoftRocks. John has taken his console and embedded it in *MPK20,* Sun's 3D virtual collaborative working environment[31]. A snapshot of jsdr in action is shown in Illustration XX. MPK20 is described as “Second Life for Professionals,” and as such it exhibits many of the necessary features for realizing our kind of VR-SDR interface.

I'm personally not so concerned with having avatars strolling around inside my virtual radio, though some may feel differently. What we're more concerned with here is the machinery that makes MPK20 tick -- the various algebraic transformations of the UI components as they're steered around in three dimensions, and how the controllers they're linked with can be instantiated as distinct, tangible virtual objects as well. The typical repertoire of 3D transforms – translation, rotation, dilation, shrinkage, warping, all in three dimensions – are what we need to build dynamic UI devices for complex parallel radios with multiple overlaid controllers. What John shows us is, pointedly, how direct the job is to realize them by falling back on a rich 3D virtual environment.


Applications Places System
Project Wonderland
File View Shared Apps Window Tools Help
File Edit View Terminal Tabs Help
SDR-1000
-81 dBm
TUNING CONTROL
RIT 7.065200
XIT 7.056000
A>B A<B A<>E SPLIT
LSB USB DSB CW/L CW/U FM/N AM SAM DIGU SPEC DIGL
UP 100 DOWN
TUNING TX Contr DISP/DEC FILTER AUDIO CW TX MEM
setup
ON Exit
SPECTRUM PANADAP SCOPE PHASE PHASE2
Project Wonderland
Performance
Users

```
[jm57878@localhost lg3d-wonderland]$ cd ..
```

Illustration 2: A VR-SDR in a VR World.

In our case, I suspect, the rich 3D environment will be not a virtual collaboration space, but a 3D compositing window manager like Beryl[32] or Compiz[33] (the two have recently reunited as a single project).

31 http://research.sun.com/projects/mc/mpk20.html
32 http://www.beryl-project.org
33 http://compiz.org

The following series of illustrations might help to show how many of the display and UI features in Roger's Dream Station are more properly OS and window manager capabilities.

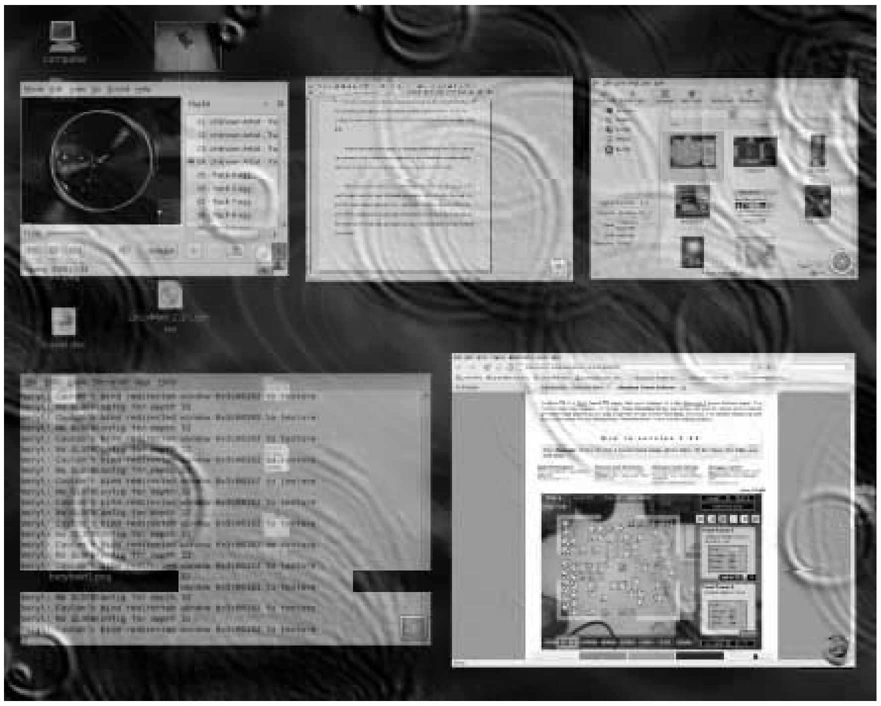


Illustration 3: Compositing Window Manager Arranging with Effects.

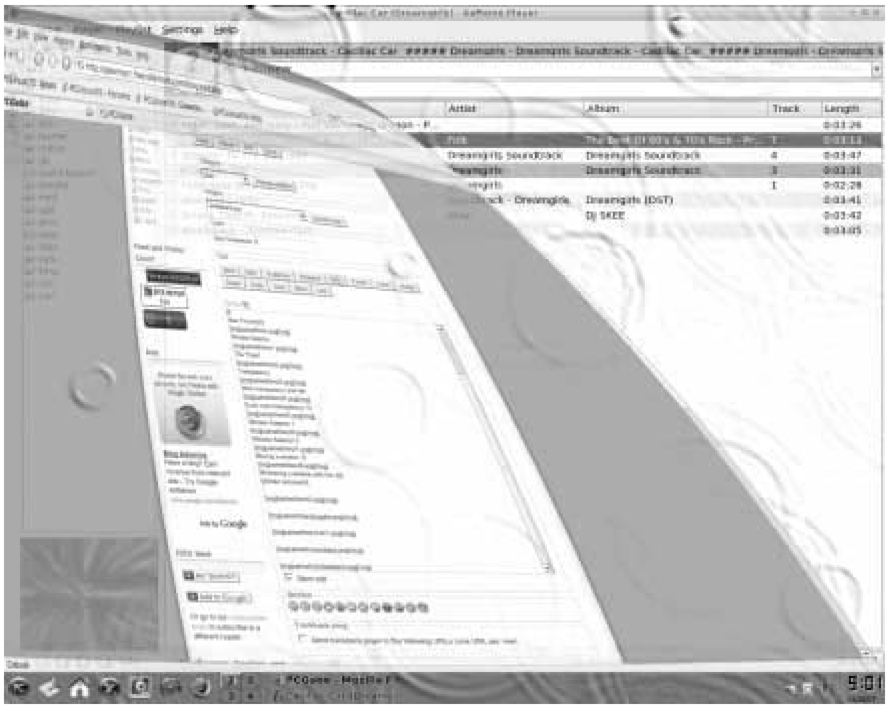


Illustration 4: Beryl dynamic effects.

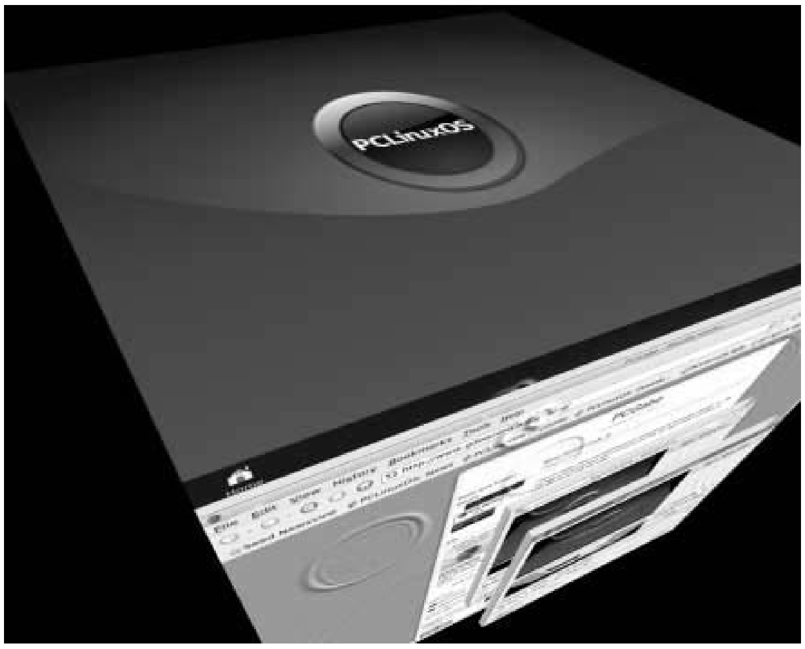


Illustration 5: A Beryl Desktop Cube showing stacked overlaid windows.

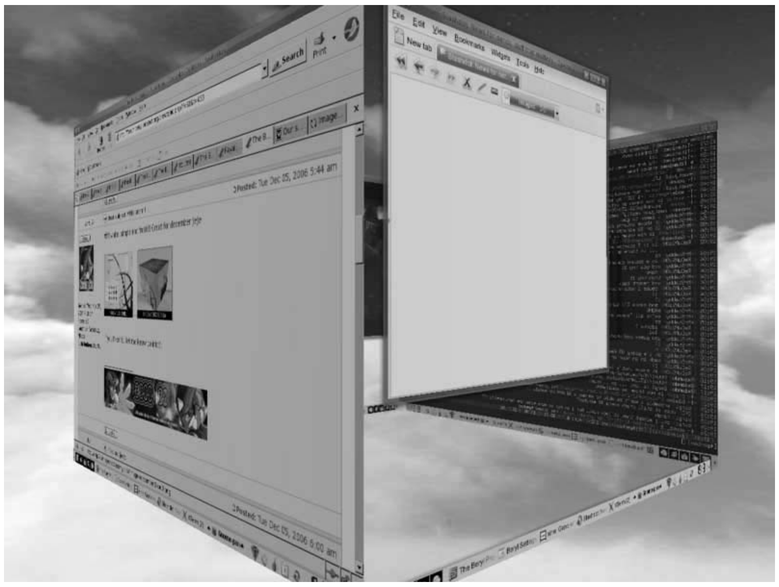


Illustration 6: Beryl Desktop Cube with Overlaid Windows.

Acknowledgments, and a Postscript Concerning Formalism

I need to acknowledge the contributions of quite a few people to the development of the thinking on these topics, such as it is. First of all, many of the ground requirements have been established by three great microwavers, Duane Grotophorst, N9DG, Phil Lanese, K3IB, and of course Roger Rehr, W3SZ. Eric Blossom, K7GNU and Matt Ettus, N2MJI, the principals behind GnuRadio, have been forceful activists in building a practical infrastructure for SDR in general. Dave Bernstein, AA6YQ, has been most helpful as an informed and skeptical observer. My friends Scott Alexander, AB2WF and Andy Stillinger, WA2DKJ have invested considerable personal time and energy. And finally, I must as always thank my wife Sandra, who pointed out how much better this essay would have been had I taken time to translate it into English.

If you're mathematically inclined, the territory circumscribed by VR-SDR – a large one, it's true – is a playground for invention. Each of the major topics of this essay – functional decomposition, the process calculus, and linear transformations – has an analogous, well-developed system of models and formal vocabulary. There are some intriguing mathematical aspects to the areas of signal processing theory that underlie software radio, and they are largely untouched as yet. While it's hard to know for sure until the theorems and proofs are there, the mathematical theory of software radio may well show the way to breathtaking improvements in algorithmic efficiency, in much the same way as linear algebra makes the visual parts of video gaming possible. Again, we can't elaborate the point much here. Nevertheless it's a topic worth a discussion of its own, which might be illuminating even in its primitive form.

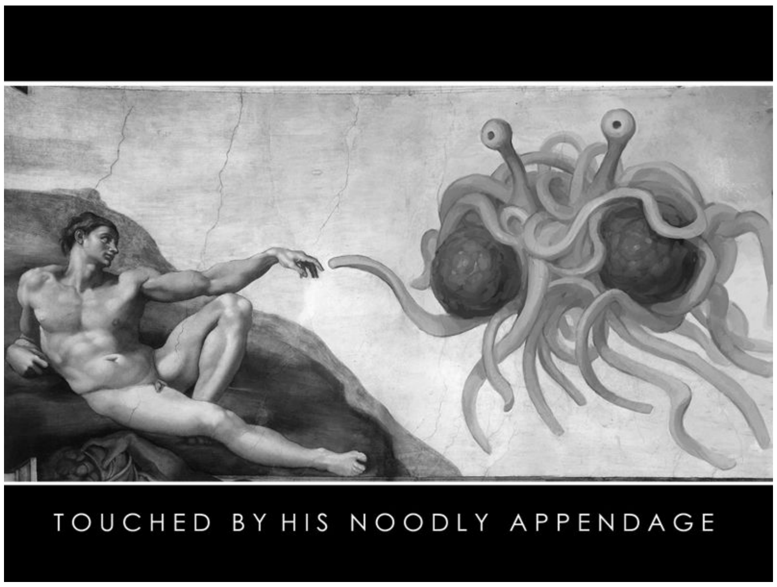


Illustration 7: The FSM at Creation Time.