EC 6014-COGNITIVE RADIO

UNIT II

Communications system is the set of devices by which one employs a communications channel to convey information to a recipient. From a radio engineering perspective, the communications device must first encode the information from the source into a suitable electronic representation. The device then must transform this internal form into a waveform compatible with the radio frequency (RF) communications channel. The channel distorts the RF signal, adds noise, and creates time-delayed distorted replicas of the signal. The process is reversed in the receiver as illustrated in Fig. 1. In this venerable historical framework, control of the radio needs little attention, generally limited to power on/off, audio volume control, a noise-riding receiver threshold, and a switch to manually select from among predefined RF channels. Several such transmitters and receivers located and working together comprise a radio node. The multi-band, multimode, multithreaded, multipersonality capabilities of software radios require an expansion of this simple model.


Technology advances have given a new level of physical-layer flexibility of the radio. Multiband technology, first of all, accesses more than one RF band of communications channels at once. The RF channel of Fig. 1 then is generalized to the channel set of Fig. 2. This set includes RF channels, but radio nodes like personal communications system (PCS) base stations and portable military radios also interconnect to fiber and cable; therefore, these are also included in the channel set.

The channel encoder functions are expanded to programmable RF/channel access, intermediate frequency (IF) processing, and modem. Antennas and RF conversion that span multiple RF bands (and fiber-optic interfaces) comprise the RF/channel access function. IF processing may include filtering, further frequency translation, space/time diversity processing, beamforming, and related functions. Multimode radios generate multiple air interface waveforms (“modes”) defined principally in the modem, the RF channel modulator–demodulator. These waveforms may be in different bands, and may span multiple bands. The source and channel coders of Fig. 1 therefore become the multiple personalities of Fig. 2. A personality combines RF band, channel set (e.g., control and traffic channels), air interface waveform, protocol, and related functions.

Although many applications do not require information security (INFOSEC), there are incentives for its use. Authentication reduces fraud. Transmission security (TRANSEC) hides the fact of a communications event (e.g., by spread-spectrum techniques). INFOSEC is therefore included in Fig. 2, although the function may be null for many applications. In addition, the source coder/decoder pair of Fig. 1 must now be expanded to include the data, facsimile, video, and multimedia sources implicit in Fig. 2. Some sources will be physically remote from the radio node, e.g., connected via the synchronous digital hierarchy (SDH), a local area network (LAN), or other network through service and network support of Fig. 2.

These functions are implemented in multithreaded, multi-processor software orchestrated by a joint control function. Joint control assures system stability, error recovery, timely data flow, and isochronous streaming of voice and video. As radios become more advanced, joint control becomes more complex, evolving toward autonomous selection of band
Mode and data format. Any of the functions may be singleton (e.g., single band versus multiple bands) or null, further complicating joint control. Joint control integrates fault modes, personalities, and support functions on a limited resource of applications-specific integrated circuits (ASIC’s), field-programmable gate arrays (FPGA’s), DSP’s, and general-purpose computers to yield a reliable telecommunications object.

A software radio can upload new air interface personalities. These may modify any aspect of the air interface, including whether the waveform is hopped, spread, or otherwise constructed. The required resources (e.g., bandwidth, memory, and processing capacity), of course, must not exceed those available. Some mechanism for evolution support is therefore necessary to define the waveform personalities, to download them (e.g., over the air), and to assure that each new personality is safe before being activated. To type certify such a radio, one must guarantee that the properties specified by the regulatory bodies are preserved in spite of this high degree of flexibility. The need for such guarantees motivates the study of the mathematical properties of this architecture.

Traditionally, one may model the statistical demand for computational resources versus processing capacity using queueing theory. Real-time performance can be assured in a fixed architecture using this approach. Plug-and-play, however, creates a variable architecture as modules are introduced into the environment and removed. This raises the complexity of the statistics, particularly in complex nodes such as a base station cell site in which hundreds of users can invoke dozens of variable-bandwidth services in a pool of shared DSP’s. A deeper understanding of the statistical properties of such environments is no doubt needed. Underlying such statistical analyses, there must be a predictable computational-demand relationship between the plug-and-play module and the environment. This calls for a theory of plug-and-play resource bounds for the software radio within which such predictable relationships will exist.

An obvious challenge is to define interface points for plug-and-play hardware and software
modules. Industry organizations including the Software-Defined Radio (SDR) Forum (formerly the Modular Multifunction Information Transfer Systems-MMITS Forum) are in the process of identifying such interface points using generalized applications programmers interfaces (API’s). A less obvious challenge is to define architecture principles that assure that plug-and-play architectures will have the mathematical properties of controllability and predictability necessary for true plug-and-play services.

1.1 Architecture Goals

A successful plug-and-play architecture entails at least the following.

*Compatibility:* The structure of plug-and-play modules must be compatible with that of the software radio environment—arcs plug into nodes for function composition. *Predictability:* Module composition must preserve radio service-defining properties of the system, and when control is exerted, it must not have unintended consequences. Modules must not consume excessive computational resources.

An architecture is a framework in which a set of components is used to achieve specified functions (services) within specified constraints or design rules. Compatibility and predictability are the properties of plug-and-play architecture. A mathematical framework for a plug-and-play architecture should inductively establish the desired properties of a given set of components within specified design rules. Due to the open-ended nature of radio services and technology, the framework must be extensible both to new services and to new implementation platforms.

2. BASIC SDR

This section defines the software radio from several perspectives: engineering design, topological structure, and computational structure. The programmable digital radio (PDR) is defined and differentiated from the software radio.

![Diagram of software radio functions and components](image-url)

Fig. 4. Key software radio functions and components.
The software radio imparts multiple personalities to a radio in a specific way. A software radio defines all aspects of the air interface including RF channel access and waveform synthesis in software. In the software radio, then, wide-band analog-to-digital and digital-to-analog converters transform each RF service band among digital and analog forms at IF as in Fig. 4. The wide-band digitized receiver stream of bandwidth \( W_0 \) accommodates all subscriber channels, each of which has bandwidth \( W_c \) (\( W_c < W_0 \)).

In the software radio of Fig. 4, IF ADC and DAC channels are processed isochronously using programmable digital hardware, and software. IF processing may include filtering to isolate subscriber channels, digital formation of nulls and beams, joint space–time equalization, integration of space diversity, polarization or frequency diversity channels and other means of acquiring a high-quality waveform. There may be multiple IF frequencies (in the heterodyne receiver) or the IF frequency may be zero (in the homodyne receiver). IF processing may be null, for example, in a direct conversion receiver. Digital downconversion is the process of using the frequency-domain periodicity of sampled bandpass waveforms to translate the waveform to baseband without analog heterodyning. Preselection filters with the necessary 80+ dB of out-of-band attenuation are becoming available in superconducting RF filters. Local oscillator leakage may also be reduced proportionally so that digital downconversion is becoming more feasible. The mathematical framework represents plug- and-play aspects of such variations via sets of arcs and nodes with subset relationships.

In the software radio transmitter, baseband signals are transformed into sampled channel waveforms via channel modem functions implemented in software to drive high-performance DAC’s. These signals may be preemphasized or nonlinearly precoded by the IF processing software. In some implementations, modem functions, IF processing, and RF channel access may be amalgamated into a single component such as a direct conversion receiver RFIC. In addition, dynamic compilation of software or real-time switching among personalities can allow these discrete functions to be integrated into a single component such as an FPGA. The mathematical framework represents both discrete and integrated implementations. In discrete implementations, each edge in the topological space is also an arc corresponding to a component. In an integrated implementation, a component arc subsumes many topological edges.

2.1. Programmable Digital Radios

There are many ways to impart more than one personality to a radio. The term “PDR” applies to those radios that use a hardware-intensive mix of hardware and software techniques to access more than one RF band with a choice of air interface modes. A PDR’s programmability may be achieved using baseband DSP’s. However, hardware modules must be interchanged for the radio to change RF band and air interface mode (“waveform”).

The software-defined radio (SDR) was defined by BellSouth to describe an evolution toward greater programmability of a wireless infrastructure. This evolution includes programmable multiband, multimode radios implemented using the PDR approach. An IS-95/AMPS handset, for example, may employ a code-division multiple-access (CDMA) chip set for IS-95; an analog mobile phone system (AMPS) chip set; a dual-mode RF integrated circuit (RFIC) chip set; a DSP chip for filtering, voice coding, and computationally intensive tasks; and a microcontroller for user interface and system control. Many of the functions are programmable, but the CDMA despreader, for example, is defined in an ASIC, and may not be changed in the field. Over time, functions initially implemented in hardware (e.g., the RF modem) will migrate to software and conversely. The mathematical framework must therefore capture SDR migration between hardware and software.
3. HARDWARE ARCHITECTURE

3.1. A Software-Equivalent Model of Hardware

The ADC and DAC define the point at which functions are potentially software defined. We may call this the digital access point. Relatively inflexible analog and digital hardware such as ASIC’s may be necessary in a handset, e.g., for direct conversion of the channel waveform to a bitstream. Although an ASIC may have a few control parameters, the personality of an ASIC cannot be changed in the field (e.g., from one air interface standard to another). To change the air interface personality, one must replace the ASIC, switch among different ASIC’s, or switch among modes of one ASIC. In the tradeoffs among size, weight, power, cost, and flexibility, fixed-function ASIC’s generally consume less power, take less space, weigh less, and cost less per device than the programmable DSP equivalent. Therefore, increased flexibility comes at a price. One way of representing these differences mathematically is to attribute equivalent computational capacity to the nonprogrammable devices.

In the topological model of the hypothetical radio illustrated in Fig. 5 (top), an RF ASIC (A) transforms the analog signal from the antenna directly to input for the baseband modem. The software-equivalent model (B, bottom) postulates an RF ADC with exactly the bandpass characteristics of the RF ASIC. The frequency translation behavior of RF ASIC (A) is modeled as the frequency translation software process of model (B). The filtering behavior of the RF ASIC is modeled by the channel filter software of (B). The sequence of RF ADC, frequency translation, and channel filter arcs constitute the software-equivalent model of this ASIC. The equivalent computational capability of RFIC (A) is the total computational demand of the postulated software processes (B). Since analog hardware has variable performance over time and over different devices, the exact filter model of a given RF ASIC will change over time. Thus, each device is represented in a set of models.

Fig. 5. (A) Topology of an RF ASIC and (B) its digital equivalent.
3.2. Quantifying Degrees of Programmability

The degree of programmability of an implementation is fundamental to software radio architecture. Since contemporary radios use a mix of processor types, one must characterize this mix precisely. Consider the highest level topological model of a radio (a single arc). This arc may be hierarchically divided into its primitive components. Hardware primitives are the discrete devices, while software primitives are single machine instructions. Primitives that may be redefined via software in the field are labeled. The number of labeled primitive arcs divided by the total number of primitive arcs is a measure of programmability. Since an ASIC’s programmability is limited to the modification of a few parameters, most of its gate-level arcs will not be labeled.

FPGA’s are, in principle, completely programmable. In practice, they are more programmable than ASIC’s, but subject to gate and interconnect constraints. Programmable radios have been based almost entirely on reconfigurable FPGA’s. The field programmability of an FPGA is more constrained than that of a DSP chip, in part because of the likelihood of running out of usable gates on the FPGA as radio functions expand. The topological model of each type of device allows one to characterize programmability.
more precisely. In the FPGA topology of Fig. 7, two dimensions (corresponding to states and interconnect) represent the allocation of functions to hardware.

Suppose that an advanced timing recovery algorithm, comprising, say, 10% of the FPGA area is to be downloaded to the radio. As shown in the figure, its needs are incompatible with the gate use of the existing timing recovery logic. Assume that one redefines the personality of the FPGA to accommodate the new logic. The download size increases from the 10% needed for the increment to 100% for the entire personality, a 900% increase in the size of the download. It is also possible that a moderately populated (70%) FPGA will be unable to accommodate the 10% download because of hardware constraints such as the required placement of I/O buffers, lack of registers, etc. A similar topological model of a DSP chip is shown in Fig. 8. The additional 10% of DSP code associated with the advanced timing recovery logic (now implemented in software) is accommodated via space available in the nonvolatile random access memory (RAM). The memory map allocates this logic to RAM hardware.

DSP may appear easier to program than FPGA’s because RAM allocation is accomplished by loader software while gate allocation in an FPGA generally requires an experienced designer. New waveforms also seem to outgrow the gates of an FPGA more easily than they outgrow the program memory of a DSP chip. On the other hand, the timing constraints of DSP software are more severe than the timing of an equivalent FPGA. Logic in an FPGA runs at nearly the chip’s clock rate, while the DSP code runs from one to three orders of magnitude slower than the system clock. The tradeoff of ASIC hardware allocation and DSP task timing is reflected in these subsets of the topological model. Complex or reduced instruction set computers (CISC/RISC) provide less hardware acceleration than DSP chips. To quantify the degree of flexibility of DSP, CISC, and RISC processors, one again defines an appropriate topological space. Let \( \{ISA\} \) be the space of single-instruction register-state transformations in a processor with a given instruction set architecture (ISA). From an arbitrary initial state, a DSP has many more edges connecting reachable data states than a CISC processor, which in turn has more arcs than an RISC processor. So, from the perspective of \( \{ISA\} \) topology, the RISC processor is the simplest, and thus in some sense, the most general programmable platform. CISC is more complex, and DSP the most complex of the fixed ISA machines.

Performance of high-quality code underscores these differences. DSP code that employs zero-overhead loops with full register stacks and processing elements yields higher throughput than CISC code of a processor with the same system clock, memory, and I/O delays. FPGA’s, on the other hand, effectively have a variable ISA that makes them even more computationally efficient than DSP machines, with an attendant loss of flexibility. That is, once we employ an FPGA’s gates for a specific task (e.g., a digital filtering algorithm), those gates in that configuration cannot be used for a different task (e.g., a timing recovery algorithm). A reconfigurable FPGA’s gates may support both functions, provided reconfiguration delays are tolerable. DSP chips use the same gates for both functions time shared through software.

These relationships define a phase space for the software radio. Physicists use a phase space to represent states of a substance (e.g., solid, liquid, and gas) as a function of parameters such as temperature and pressure. The software radio phase space of Fig. 9 represents the states of radio implementations as a function of the digital access point and the degree of programmability.

Fig. 9 places five types of radios in the phase space. The vertical axis represents the bandwidth at the digital access point. The horizontal axis represents the degree of flexibility, the fraction of functionality that may be changed in the field using plug-and-play software. To place a system in the phase space, one examines the ADC’s and DAC’s, placing the digital access point where the functionality is fully programmable. One places the radio on the horizontal axis according to the aggregate degree of programmability of the device.
The HF STR-2000, a commercial product of Standard Marine AB, shown at point employs baseband DSP using the TMS320C30. Most commercial off-the-shelf (COTS) cellular telephone handsets fall near .

ASIC’s provide much of the equivalent processing capacity, shifting these designs toward the less programmable end of the axis. Current digital cell site designs, similarly, rely heavily on digital filter ASIC’s for frequency translation and filtering, but they access the spectrum at IF. SPEAKEasy II, provides a GFLOP of programmable DSP, shifting this implementation to the right. A product of research at the Massachusetts Institute of Technology, the virtual radio delivers a single channel radio using a general-purpose processor, DEC’s alpha. This is the most general-purpose computing platform reported in the literature. Point represents the ideal software radio with the digital access point at RF and all functions programmed on RISC processors. Although maximally flexible, and thus of research interest, such designs tend to be economically impractical. This phase space quantitatively differentiates software radios in the upper right quadrant from the PDR’s elsewhere in the figure.

A PDR is not a software radio if any crucial aspect of the channel waveform is implemented using programmable hardware (such as a voltage-controlled oscillator) rather than using software (e.g., sin/cosine lookup table). A joint tactical information dissemination system (JTIDS) radio simple-mentation, with a 3 MHz IF from which hop frequencies are generated using a 250 MHz programmable local oscillator, is not a software radio for that waveform. On the other hand, a 250 MHz digitized IF that could set every hop frequency in software would be a software radio. The radio may meet the software radio criterion for a large class of narrow-band waveforms, while failing for wide-band waveforms. The phase space of the topological model thus unambiguously defines the degree to which an implementation is reprogrammable. Given an unambiguous definition, one may consider the components in more detail

![Software radio phase space diagram]

Fig. 9. Software radio phase space.
TABLE I ATTRIBUTES OF TOP LEVEL SOFTWARE RADIO FUNCTIONAL COMPONENT

<table>
<thead>
<tr>
<th>Functional Component</th>
<th>Attributes</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Source Coding &amp; Decoding</em></td>
<td>Audio, video, fax and data interfaces</td>
<td>Ubiquitous standard algorithms (e.g. ITU[1], ETSI[2])</td>
</tr>
<tr>
<td><em>Service &amp; Network Support</em></td>
<td>Multiplexing; setup and control; data services; internetworking</td>
<td>Wireline and Internet standards including mobility [3]</td>
</tr>
<tr>
<td><em>Information Security</em></td>
<td>Transmission security, authentication, non-repudiation, privacy, data integrity</td>
<td>May be null, but is increasingly essential in wireless applications [4]</td>
</tr>
<tr>
<td><em>Channel Coding &amp; Decoding: Modem</em></td>
<td>Baseband modem, timing recovery, equalization, channel waveforms, predistortion, black data processing, etc.</td>
<td>INFOSEC, modem, and IF interfaces are not standardized</td>
</tr>
<tr>
<td><em>IF Processing</em></td>
<td>Beamforming, diversity combining, characterization of all IF channels</td>
<td>Innovative channel decoding for signal and QoS enhancement</td>
</tr>
<tr>
<td><em>RF Access</em></td>
<td>Antenna, diversity, RF conversion</td>
<td>IF interfaces are not standardized</td>
</tr>
<tr>
<td><em>Channel Set(s)</em></td>
<td>Simultaneity, multiband propagation, wireline interoperability</td>
<td>Automatically employ multiple channels or modes for managed QoS</td>
</tr>
<tr>
<td><em>Multiple Personalities</em></td>
<td>Multiband, multimode, agile services, interoperable with legacy modes</td>
<td>Multiple simultaneous personalities may cause considerable RFI</td>
</tr>
<tr>
<td><em>Evolution Support</em></td>
<td>Define &amp; manage personalities</td>
<td>Local or network support</td>
</tr>
<tr>
<td><em>Joint Control</em></td>
<td>Joint source/channel coding, dynamic QoS vs. load control, processing resource management</td>
<td>Integrates user and network interfaces; multi-user; multiband; and multimode capabilities</td>
</tr>
</tbody>
</table>

* Interfaces to these functions have historically been internal to the radio, not plug-and-play

Fig. 10. Topological model of dual band handset streams
Radio components are represented as arcs in the topological model. An arc may be a union or composition of other arcs, defining a natural encapsulation hierarchy for the radio system. At the top level of the hierarchy (the “context” level in object-oriented design), the radio node is a black box mapping air interface, user, and network events to appropriate responses. The functions of Fig. 2 define the second level of partitioning with the component attributes highlighted in Table I.

The attributes listed in this table indicate the allocation of functions to top-level components. In a topological space, each allocated function may be a singleton, a subset, or the null set. This corresponds to the occurrence of a function implemented in a single component, implemented multiple times (e.g., in parallel channels) in multiple components, or not implemented at all. The remarks column highlights functional diversity leveraged by the software radio architecture.

The stream-oriented functional components are source coding and decoding, service and network support, INFOSEC, RF/channel modem, IF processing, and RF/channel access. Each is represented topologically by a pair of arcs between data interfaces, the topological domain, and range of the maps. A topological model of a specific radio consists of those arcs that correspond to the components. For example, the signal-stream topology of a dual-mode handset is shown in Fig. 10. The critical interfaces among dual-band antenna, IF ADC, baseband modem, and vocoder of this notional device are readily apparent in the topological model. Additional interfaces associated with the functions of Table I are shown in Fig. 11. Analog (audio and video) waveforms comprise the interface between the source set and the source coding and decoding functions. Source bits are encoded, but services and network support adds forward error control structure. If INFOSEC is null, protected bits become clear bits. Interfaces may be null at one level and visible at another. An RFIC, for example, may subsume the IF waveform interface, exhibiting only baseband and RF waveforms in the higher level of the topological space. Implementation topologies are constrained by these in-terfaces and related standards as summarized in Table II. Interfaces are represented in a space which includes the interface signal itself $x(t)$ and related meta-level characteristics: implementation class (hardware or software), impedance, connector type, bandwidth, etc. The analog stream has infinite dimensionality in both time and amplitude, defined over $\mathbb{R}$. A digitized analog stream consists of an infinite stream of one-dimensional vectors, the samples. But, given a pair of sample buffers of length $N$, this stream is reduced to dimension $N$. 

![Diagram](image-url)
In addition, as shown in Fig. 12, the topological model structures the meta-level aspects of these interfaces. Each dimension of the space must be identified, and the associated elements that apply must be specified. The syntactic and semantic definition of the topological spaces of such a generic radio would constitute a knowledge representation language (KRL) [47] for radio (an RKRL). The definition of a complete, extensible RKRL is a significant undertaking that is beyond the scope of this introduction. Important properties of plug- and-play interfaces may be defined without a complete RKRL, however. In particular, general properties of the interfaces are given in the following table II,

<table>
<thead>
<tr>
<th>Interface</th>
<th>Key Characteristics</th>
<th>Topological Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Stream</td>
<td>Audio, video, facsimile streams</td>
<td>Infinite dimensional; Filtering constraints comprise open ball(s)</td>
</tr>
<tr>
<td>Source Bitstream</td>
<td>Coded bitstreams and packets. ADCs define a finite window into a quantized, discrete-time sampled waveform.</td>
<td>Finite dimensional; frame and data structure defines subspaces. Finite precision defines a Dynamic Range Subspace for the ADC</td>
</tr>
<tr>
<td>Clear Bitstream</td>
<td>Framed, Multiplexed, Forward Error Controlled (FEC) bitstreams and packets</td>
<td>Finite dimensional; FEC subspaces have rich algebraic properties</td>
</tr>
<tr>
<td>Protected Bitstream</td>
<td>Random challenge, authentication responses, public key; enciphered bitstreams and packets</td>
<td>Finite dimensional; randomized streams; complex message passing for downloads; If null, interface reverts to clear bits</td>
</tr>
<tr>
<td>Baseband Waveform</td>
<td>Discrete time synchronous quantized sample streams (one per carrier)</td>
<td>Digital waveform properties determine fidelity of analytic representation</td>
</tr>
<tr>
<td>IF Waveform</td>
<td>Composite, digitally pre-emphasized waveform ready for up-conversion</td>
<td>Analog IF has infinite dimensional topology; Digital IF may have baseband product topology</td>
</tr>
<tr>
<td>RF Waveform</td>
<td>Power level, shape, adjacent channel interference, etc. are controlled</td>
<td>Analog RF has infinite dimensional topology; Includes spatial and temporal dimensions</td>
</tr>
<tr>
<td>Network Interface</td>
<td>Packaged bitstreams may require ATM, SS7, or ISO protocol stack processing</td>
<td>Synchronous Digital Hierarchy (SDH), Signaling System 7 (SS7) subspaces</td>
</tr>
<tr>
<td>Joint Control</td>
<td>Control interfaces to all hardware and software; initialization; fault-recovery</td>
<td>(Not illustrated in the figure) Parameter spaces; non-linear logic subspaces</td>
</tr>
<tr>
<td>Software Objects</td>
<td>Download from evolution support systems</td>
<td>Represents binaries, applets; includes self-descriptive language subspaces</td>
</tr>
<tr>
<td>Load/Execute</td>
<td>Software object encapsulation</td>
<td>Download topologies are highly nonlinear</td>
</tr>
</tbody>
</table>

**TABLE II TOP-LEVEL INTERFACE TOPOLOGIES**
computational components can be studied independently of an RKRL. In addition, air interface specifications and radio API’s constitute informal RKRL’s. Such interface domain descriptions could promote the assured delivery of services in the software radio’s plug-and-play environment. A plug-and-play interface should effectively separate the module from the rest of the system. Effective separability implies at least the following topological properties.

1) Composition of module functions on the interface topology yields a well-defined system-level function consuming specified resources to deliver the intended service.
2) Performance (e.g., spectral purity, data formats, through-put, response time, etc.) under function composition is within specified bounds.

The separability of modules at plug-and-play interface points is illustrated in Fig. 13. Plug-and-play modules may comprise top-level functional components such as the modem. Or, they may be defined across arbitrary points deep in the hierarchy such as at the vocoder.

5. COMPUTATIONAL PROPERTIES OF FUNCTIONAL COMPONENTS

COMPUTATIONAL PROCESSING RESOURCES IN SDR (NOV/DEC 2016, NOV/DEC 2017)

A fundamental aspect of software radio architecture is the set of conditions under which plugging in a module results in acceptable use of computational resources. The composability of functions defined over general topological spaces is well understood. Homeomorphisms are topology-preserving maps that may be composed to yield other homeomorphisms provided the domain of one map and the range of the other are topologically compatible. From a topological perspective,
the domain of software consists of those inputs over which its results are defined. The range consists of the corresponding results (including side effects and returned values). Under what conditions can well-behaved software be composed with other well-behaved software to yield a well-behaved system? This very general question may be undecidable. But software radios are engineering systems with timing constraints that allow us to prescribe constraints on the topological structure of the software, and to establish conditions under which the composition of software modules consumes finite resources that can be predicted accurately.

The concept in addressing bounded resource use is to construct the largest set of functions for which predictable finite resource consumption may be guaranteed. In addition, the composition of such functions must also be bounded for a proof that defines conditions under which plug-and-play modules will not use excessive resources. Furthermore, any constraints on the radio functions that can be computed with such functions must be specified. The analysis of this section shows that the bounded recursive functions both guarantee predictably bounded resource use and present no constraints on radio functionality. A system constructed using these functions has the ability to maintain predictable throughput across dynamic changes of plug-and-play software modules.

A. Models of Computation

- Each ASIC and FPGA are attributed the computational capability of a canonical processor, which may in turn be modeled as a (collection of) von Neuman machine(s). Each procedure encoded in software on a von Neuman processor may be modeled in terms of computability and concrete complexity as a random access machine (RAM).

- A RAM consists of a state machine, input, and output arrays; internal registers; the capability to load and store data via direct and indirect addresses; and the capability to increment memory values. The RAM model provides an intuitive but mathematically precise description of a single von Neuman processor.

- RAM model has also been proven equivalent to the recursive function and Turing machine models of computing. Important theoretical properties include the ability to simulate in polynomial time capabilities that require exponential time on Turing machines. Let \( \text{RAM} \) represent the RAM model of the instruction set architecture (ISA) of an arbitrary processor. Theorems outlined below use the recursive function and The RAM models to establish tight upper bounds on processing resources, a necessary condition for predictable throughput.

B. Primitive Recursive Functions
The recursive function model of computing consists of a set of functions from $\mathbb{N}$, the natural numbers, onto $\mathbb{N}$ with closure properties that define classes of functions. The *primitive recursive functions* consist of the following.

1) The *zero* function $\mathit{z} : \mathbb{N} \to \mathbb{N} : \mathit{z}(x) = 0$, which yields the constant zero for any natural number $x$ in $\mathbb{N}$.

2) The *successor* function $\mathit{s} : \mathbb{N} \to \mathbb{N} : \mathit{s}(x) = x + 1$, which associates a successor to each $x$ in $\mathbb{N}$.

3) The *projection* function $U^m_n : \mathbb{N}_n \to \mathbb{N} : U^m_n(x_1, x_2, \cdots , x_n) = x_m$ by which arguments may be selected, with closure under:

4) *Composition*: for $g_1, g_2 \cdots g_m : N^n \to N$ and $h : N^m \to N$, $f(x_n) = h(g_1(x_n), g_2(x_n) \cdots g_m(x_n))$, where $(x_n)$ is brief notation for $(x_1, x_2 \cdots x_n)$; equivalently, $f = h^*(g_m)$.

That is, the set is closed under composition of functions. One can compose functions in a natural way that corresponds to function calls and/or selection of data inputs in a random access machine.

5) *Primitive Recursion*: for $g : \mathbb{N}^n \to \mathbb{N}$, $h : \mathbb{N}^{n+2} \to \mathbb{N}$, and finite $y > 0$, $f(x_n, y) = g(x_n)$ if $y = 0$, and

\[ h(x_n, (y - 1), f(x_n, (y - 1))] \quad \text{for } y > 0. \]
That is, the set of functions is closed under primitive recursion. Primitive recursion for \( y = 1 \) and \( h(x_n, 0, f(x_n, 0)) = h(x_n) \), \( f \) is simply the conditional execution of \( g \) or \( h \) based on the value of \( y \) [i.e., since \( f(x_n, 0) = g(x_n) \), but \( f(x_n, 1) = h(x_n, 0, g(x_n)) = h(x_n) \)]. This recursive pattern is equivalent to the if–then–else programming construct in terms of effective computability. Since \( f \) appears in its own definition, it requires a pushdown stack of successive arguments. But since initially \( y > 0 \) and \( y \) decreases at each invocation, the procedure will terminate, and a stack which is as large as \( |y| \) will not overflow. It is assumed that the stack allocator in a specific \{ISA\} keeps the finite stack from overflowing, provided an upper bound for \( |y| \) is known in advance. The knowledge of such bounds is a feature of the software radio application of this theory.

The *primitive-recursive functions* are the smallest set of such functions closed under composition and primitive recursion. Sequential logic, transversal and recursive filters, bit manipulation, and data packing are common primitive-recursive functions of software radios. They include addition, subtraction, multiplication, and others that do not require iterative search.
C. The Total Recursive Functions

Iterative loops are not primitive recursive, but they are essential to radio software. The simplest model of iteration is bounded minimalization.

6) Bounded Minimalization: Let \( g: \mathbb{N}^{n+1} \rightarrow \mathbb{N} \) be primitive recursive; then

\[
f: \mathbb{N}^{n+1} \rightarrow \mathbb{N}: f(x, y) = \mu z \{ z < y \} [g(x, z) = 0]
\]

is also primitive recursive.

Read \( \mu z \) as \( f(x, y) \) is the least \( z \) less than \( y \) for which \( g(x, z) = 0 \). The class of functions closed under composition, primitive recursion, and bounded minimalization is total. That is, they are defined for all \( \mathbb{N} \). The Fortran do loop exemplifies bounded minimalization. Ackerman’s function is defined everywhere, but uses resources faster than any known function due to the structure of its recursive calling sequences. It is not considered primitive recursive, although it is a total function. Yet, for small finite parameters of Ackerman’s function which are known in advance, one can determine whether the function will exceed an upper resource bound by using a suitable step-counting function. Any primitive recursive function induces a primitive-recursive step-counting function.

- Relevance: Software radio incorporates isochronous streams for which there is a fixed timing window during which specific functions must be accomplished in order for services to be delivered properly and/or for the system to remain stable. Modules whose structure is not constrained as above may use resources that are bounded on test cases, but that “blow up” in other conditions. All primitive recursive functions can be guaranteed to either: 1) complete within a specified window or 2) be easily computed in advance to be incapable of meeting that timing window

- (Lemma 1). If the system control algorithm predicts such a fault, it can signal the network or the user and preclude the (plug-and-play) module from being invoked. Such predictable timing properties provide the foundation necessary to allocate computational resources to plug-and-play modules safely.

- Queueing theory translates service time measured in isolation into statistical bounds on isochronous performance in the service environment where other tasks are sharing the same processor(s). The control algorithm may employ such techniques to establish a statistical bound on aggregate
resource use which, if exceeded, could degrade services. But since the resource bounds may be computed, the system should not become unstable.

D. Bounding the Partial Recursive Functions

Bounded primitive recursive functions do not express all of the programming constructs needed in software radios. Notably absent are the **while** and **until** loops. These loops search for a condition under which to terminate. This condition may never occur, so the RAM programs may loop forever. Hardware and software processes that wait or search for a condition that may not occur are computationally equivalent to while or until loops. The source code of software radios like SPEAKeasy I reveals the relatively widespread use of wait-prone constructs, e.g., waiting for sufficient signal strength to initiate receiver processing. The process-dispatch loop of the kernel operating system may consume infinite resources to deliver resources to applications software. Other infinite loops are computationally equivalent to the partial recursive functions. The computational structure is called (unbounded) minimalization.

**Theorem 1 (Primitive Bounded Resources):** Any bounded primitive recursive function is equivalent to a RAM program which terminates in a finite number of steps which can be bounded tightly from above, given bounds $y_i$, of the minimalizations from which the composite functions are created.

**Lemma 1 (Primitive Bounded Execution Time):** The dedicated-processor RAM-equivalent of any bounded primitive recursive function executes in an amount of time that may be tightly bounded. One determines the maximum execution time of each class of RAM instructions on a specific ISA. The execution time bound is the product of the number of steps from Theorem 1 times the maximum execution time for that class of instruction. The time used by the function tested in isolation on a specified dedicated machine is thus an alternative step-counting function. From a software engineering perspective, this lemma states that the bounds allocated may be specified in terms of execution time on a dedicated processor.
Theorem 2 (Unbounded Loops): Software radio functions implemented with while and/or until loops or their equivalents (e.g., implemented using go-to programming styles) cannot be guaranteed to use bounded computational resources.

Outline of Proof: Construct a while loop that simulates minimalization:

While(\text{NOT}(g(x_n, y) = 0), \text{increment } y); \text{ Return } y;

One may show the equivalence of until, while, and for loops to unbounded minimalization. They thus may be undefined, and are not total. The equivalent RAM procedure may consume unbounded computational resources, “crashing” the system.

Theorem 3 (Local Feasibility of Bounded Partial Recursion): A single-threaded single processor software radio function may be synthesized in RAM instruction sequences equivalent to a resource-constrained subset of the partial recursive functions.

Outline of Construction (Locally Bounded Partial Recursion): This proof builds on Theorems 1 and 2. Unconstrained While and Until loops and their “go-to” equivalents which cannot be guaranteed to terminate are precluded. Each such construct from a conventional implementation is allocated a maximum number of iterations (or, equivalently, a maximum processor time). These limits are coded into bounded-while or bounded-until loops as indivisible operations (in the same way one that a semaphore is an indivisible operation). Hardware that waits until a condition is met is similarly bounded, redesigned to generate a fault interrupt if the condition is not met after a specified time interval. Watchdog timers exemplify this principle.
Theorem 4 (Totality): The bounded partial recursive functions are total.

Outline of Proof: Total functions are those that are defined for all $\mathbb{N}$. Each of the primitive recursive functions is defined for all $\mathbb{N}$. The closure under composition, primitive recursion, and bounded minimalization is defined for all $\mathbb{N}$. Bounded minimalization is defined for all $\mathbb{N}$ as follows. Associate with each range of arguments of a minimalization a step-counting function with a limit $y_{\max}$. If the search terminates in fewer than $|y_{\max}|$ steps, the loop yields the result of the minimalization. If not, then the result is FAIL for all $N > |y_{\max}|$. Bounded minimalization is therefore total. The set of functions closed under such operations is therefore total.

Theorem 5 (Global Bounded Recursion): Multithreaded multiprocessor software radio functions implemented using RAM sequences that are equivalent to the bounded recursive functions may be guaranteed to run to complete or to cause a resource fault. The number of instructions before a fault may be tightly bounded using polynomial resources to compute the bound.

Outline of Proof: (Existence of Globally Bounded Recursion): Nielson and Nielson present a process algebra for a polymorphic subset of concurrent ML, a parallel multiprocess language. With this, they show that the number of processes and interprocessor communications channels associated with a subset of concurrent ML is finite. They point out that their analysis is not complete: some programs with finite resource use will be rejected as potentially infinite due to an inability to infer the finiteness of sequences of (unbounded) recursions which terminate. In addition, their semantics admit sequences in which an infinite number of tasks terminate before a finite number begins, again resulting in an erroneous rejection of a finite resource use case.
A successful plug-and-play architecture entails at least the following.

**Compatibility:** The structure of plug-and-play modules must be compatible with that of the software radio environment—arcs plug into nodes for function composition. **Predictability:** Module composition must preserve radio service-defining properties of the system, and when control is exerted, it must not have unintended consequences. Modules must not consume excessive computational resources.

An architecture is a framework in which a set of components is used to achieve specified functions (services) within specified constraints or design rules. Compatibility and predictability are the properties of plug-and-play architecture. A mathematical framework for a plug- and-play architecture should inductively establish the desired properties of a given set of components within specified design rules. Due to the open-ended nature of radio services and technology, the framework must be extensible both to new services and to new implementation platforms.

6. INTERFACE TOPOLOGIES AMONG PLUG-AND-PLAY MODULES

Constraints on the computational structure of modules can ensure bounds on resource use for plug-and-play. The following are the possibilities

**A. Topological Spaces**

- **Definition (Topological Space):** A topological space, denoted \((X, O_x)\), is a set and a family of subset \((X, O_x)\) the open sets, which include and the empty set \(\emptyset\), and which are closed under countable union and finite intersection. The topology is the family of subsets \(O_x\) which has the geometric and algebraic structure.

In a topological space, one can represent the geometric properties of interfaces among software radio modules. An interface to an analog source, for example, may be modeled as a subset of functions on \(R\) that obeys certain constraints. The constraints include bandwidth, adjacent channel interference, and minimum and maximum transmitted power. An uncountable number of such waveforms is possible in an analog interface, but regulatory bodies and the hardware limit the waveform to a structured subset of the possible waveforms. Since the space of randomly perturbed signals is also a metric interface.
B. Finite Interface Topologies

If a set \( X \) has a finite number of elements \(|X|\), all subsets are open sets (and are also closed sets). If \(|X| = M\), then the number of topologies that induce a topological space on \( X \) is \( 2^{2^M-2} \), a double exponential. Not all of the candidate topologies satisfy closure under union and finite intersection as required for a topological space. The huge number of possible topologies compels one to define finite interface topologies more compactly.

**Definition (Basis):** A set \( B \subseteq X \) is a basis for \( O_x \) if the members of \( O_x \) are the union of members of \( B \). A basis is a smaller set than \( O_x \) from which \( O_x \) may be induced by taking unions.

From a hardware perspective, a pin in a connector is an interface point \( x_i \in X \). The set \( \{X = \bigcup N \times x_i, \phi\} \) is a basis for \( O_x \). The set \( Y = \{\bigcup N-1 x_i, \{x_1\}, \phi\} \) which contains three subsets \( \{x_1\}, \phi \), and the union of all the other interface
**C. Function-Call Parameter Topologies**

The geometric structure of interface spaces may be better understood using additional notions.

*Simplex:* A simplex is an ordered set of points in a topological space that are adjacent in some sense, such as sharing a relation $R$. Higher dimensionality simplexes induce lower dimensionality simplexes. Simplexes may be embedded in Euclidean space.

*Complex:* A simplicial complex is a union of simplexes that includes the union of all of the lower dimensionality simplexes of a given simplex.

*Q-Connected:* Simplicial complexes that share a $q + 1$ face are $q$-connected.

The three vertices of a plane triangle $[A, B, C]$ comprise a two-dimensional simplex, adjacent in the sense that they are connected by the points in the plane. Each line segment joining these vertices comprises a one-dimensional

![Simplicial complex consists of two simplexes.](image)

*Fig. 14 Signal plane entity for functional call parameter topologies*
D. Plug-and-Play Interface Geometry

Plug and Play module used in personal computing enables hardware and software modules from different suppliers to work together when plugged into an existing system. Hardware modules will plug-and-play if the physical interfaces and logical structure of the functions supplied by that module are compatible with the physical interfaces, allocation of functions, and other design rules of the host hardware platform. Software modules will plug-and-play if there is a comprehensive but simple interface to the host environment, and if the module offers to the environment the information that it needs in order to employ it as a resource.

The interface geometry comprises of RF section operating in Low band RF and High band RF, IF ADC, channel filter control filter interface, channel filter, new waveform vocoder and new vocoder which are represented in Plug and Play Interfaces, Specified Interface and Arbitrary Interface. The implementation decisions made during design, development, integration, and test may constrain an interface to a point in the simplicial complex of interfaces as illustrated in Fig. Interface geometry reflects design decisions. Each domain and range of a conventional interface consists of a designated point in its simplicial complex. A plug-and-play interface, however, defines an interoperable subset of the interface space. The physical interface subspaces must change as a function of the hardware in which the service is delivered. The logical interface subspaces also may have to change as a function of the software modules configured to deliver the services. To be fully extensible, plug-and-play modules have to be combined dynamically. To do this in a controlled way, the control algorithm(s) must have a way of comparing the range of one function to the domain of the next to determine whether the functions are compatible.

ARCHITECTURE PARTITIONS

A goal of a plug-and-play software radio architecture is to provide computing resources from undefined hardware modules to support as undefined software modules for undefined services. The following architecture analysis applies the computational and topological properties introduced above to begin to identify the mathematically based partitions of a software radio system.

A. SPEAKeasy I

SPEAKeasy I resulted in the software radio source code structure summarized in Table III. The as-built code has some strong features—such as real-time performance and accurate handling of timing differences between radio networks. Since an Ada implementation was mandated, the real-time executive is the Ada run-time kernel. Ada modules can be viewed as software objects. They include databases, channels,
and agents. Databases store personalities, filter parameters, lengthy chunks of compiled code, and data sets to be loaded into a personality at run time. Channels are abstractions around which modes (e.g., HAVE QUICK) are organized. A channel is supported by several Ada packages that perform the systems-level functions of RF control, modem processing, INFOSEC, and related internetworking. A channel gets the system resources (paths or threads through the system). It installs its personality on these resources to implement a mode. It then keeps track of the overall state of the processing thread that delivers the associated services. In SPEAKeasy I, lower level modules implement the personalities of the channels. They also serve as hosts for buses, manage IO processes, access timing and positioning data, and control the radio. Channels also keep track of the status of the mode, number of resources employed, volume, data rate, throughput, network parameters such as network number, and assigned timeslot(s). The following table shows SPEAKEASY I software modules.

<table>
<thead>
<tr>
<th>Module</th>
<th>Source</th>
<th>Ada Module Descriptions/ Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>At</td>
<td>(127 kB)</td>
<td>C040 interprocessor communications</td>
</tr>
<tr>
<td>BIT</td>
<td>(318 kB)</td>
<td>Built-In-Test packages, including CRC, EEPROM, PID, I/O registers, interrupts &amp; DMA</td>
</tr>
<tr>
<td>Cm</td>
<td>(1.29 MB)</td>
<td>Configuration Management</td>
</tr>
<tr>
<td>ALE</td>
<td>(125 kB)</td>
<td>ALE: Receive (Rx) and Transmit (Tx) Functions</td>
</tr>
<tr>
<td>ALE_Rx1</td>
<td>(378 kB)</td>
<td>Automatic Link Establishment (ALE) Receive Modules</td>
</tr>
<tr>
<td>Hzq</td>
<td>(645 kB)</td>
<td>Have Quick Communications Ensemble</td>
</tr>
<tr>
<td>Hzq_Ct</td>
<td>(109 kB)</td>
<td>Control Modules (Initialization, Mode Control, Errors)</td>
</tr>
<tr>
<td>Hzq_Global</td>
<td>(25 kB)</td>
<td>Globals</td>
</tr>
<tr>
<td>Hzq_Rx</td>
<td>(379 kB)</td>
<td>Receive Mode (Synchronize, TOD, Rx, Active...)</td>
</tr>
<tr>
<td>Hzq_Tx</td>
<td>(131 kB)</td>
<td>Transmit Mode</td>
</tr>
<tr>
<td>Work</td>
<td>(299 kB)</td>
<td>ALE packages &amp; specs</td>
</tr>
<tr>
<td>Hfim</td>
<td>(518 kB)</td>
<td>HF Modern Communications Ensemble</td>
</tr>
<tr>
<td>Hfim_ctrl</td>
<td>(58 kB)</td>
<td>Controls Waveform start/stop messages; RTS Events; PM Query; TX/RX Done (local);</td>
</tr>
<tr>
<td>Hfim_dc</td>
<td>(22 kB)</td>
<td>Data Control Packages, source messages error checking</td>
</tr>
<tr>
<td>Hfim_rx</td>
<td>(289 kB)</td>
<td>Receiver Bit &amp; message operations, text I/O, Rx utilities, data correlation tables, filters, queues</td>
</tr>
<tr>
<td>Hfim_tx</td>
<td>(149 kB)</td>
<td>Squelch, TX/RX mode, TX templates, RF Control, Timing</td>
</tr>
<tr>
<td>Nbg</td>
<td>(334 kB)</td>
<td>Narrowband Frequency Hopping Group</td>
</tr>
<tr>
<td>Nbg_cst</td>
<td>(49 kB)</td>
<td>State Machine, Sync Loss, TX/ RX, Waveform, PTT State…</td>
</tr>
<tr>
<td>Nbg_global</td>
<td>(105 kB)</td>
<td>Global parameters for NBG package</td>
</tr>
<tr>
<td>Nbg_hp</td>
<td>(57 kB)</td>
<td>Hop Packages – timing, data request/processing, PTT acknowledge, cryptographic processing…</td>
</tr>
<tr>
<td>Nbg_rx</td>
<td>(73 kB)</td>
<td>Receiver Packages MFSK, Preamble, Galois (FEC), Dead Bits, Flags, Bitsync, RX flush, Detect/Track</td>
</tr>
<tr>
<td>Nbg_t</td>
<td>(49 kB)</td>
<td>TX: Amplitude, Preamble fill, IQ Samples, AM on Voice, Filter, Inter-Process Communications (IPC) Messages, SSB, DSB, QAM, QPSK, Event &amp; Constraint Checking</td>
</tr>
</tbody>
</table>

B. The Hardware-Specific Partition
The hardware–software interfaces define the lowest level partition of a radio system. From a topological perspective, processor hardware connects the states of data in its registers. The set of data states reachable in one clock cycle from any initial state is the processor’s characteristic simplex. A processor with 20 32-bit address registers, cache pointers, general-purpose registers, with multiple direct memory access (DMA) registers and other registers can, in principle, assume any of 2 states.

Paths in Simplexes Induce Partitions: Sequential simplexes are -connected to each other by the clock sequence. Over time, a series of RAM instructions traces a path through a sequence of simplexes, the union of which is a simplicial complex (SC). In 1 s, a processor with a 100 MHz instruction clock with more states.
In addition, a software radio with distributed multiprocessor hardware and interconnect devices has additional connectedness among processor simplexes as illustrated.

At each clock cycle, an edge in the simplex is traversed (e.g., the solid arrow in processor 1’s simplex at time in the figure. The path traversed through a sequence of simplexes may be subsumed into a simplicial complex entity in the figure. The dotted arrow in the SC indicates the path traversed by data sent from to during clock sequence. The way in which software constrains such paths implicitly defines architecture partitions. A larger number of possible paths indicates greater built-in flexibility while topological loops in such path suggest partitions.

![Diagram of processor simplexes and interconnect simplexes](image)

**Fig. 16** Processor simplexes (S1, S2), interconnect simplexes (I), and an equivalent simplicial complex SC.

**Interrupt Service Routine (ISR) Topological Loops:** Consider hardware-dependent software such as operating system services. This software consumes processing resources in a way that is tightly coupled to hardware-related processor states. A typical ISR, for example, might turn off hardware interrupts, push the processor state onto a stack, test the interrupt condition, set a dispatch pointer, restore the registers, and exit.

**The Topology of the Kernel Substrate:** The set kernel comprises the software most closely aligned to hardware states and having the shortest topological loops. The kernel is the lowest level software substrate. Kernel sequences can be modeled as extensions to the processor’s native ISA. The hardware interrupt ISR, for example, could be modeled as the function Service() defined over the set Interrupt-Registers. Service() then becomes an instruction in kernel. From a topological perspective, Service defines a subgraph within the processor’s SC.

**C. Topology of Infrastructure Software**

Given a set of kernel software, the next step in a bottom up definition of virtual machines identifies those functions that allocate, set up, and control physical resources to create logical resources. Figure 17 lists the function calls required for distributed processing. These functions include the structures recommended in ITU’s X.900 open distributed processing reference model. They also include structures unique to radio applications such as frequency distribution. This infrastructure code manages control flow paths; signal flow paths; and timing, frequency, and positioning information. It also includes features needed for isochronous delivery of voice, video, and other real-time streams. The control flow paths mediate message passing among most objects in the system. Error logging, semaphores that manage shared resources, and bus access protocols are examples of control message flows.
D. Radio State Machines

State machines control access to many software radio resources. State machines typically control the transmit and receive channels and fault recovery actions. A resource allocation state machine is illustrated in Figure. Three states are shown in boxes: waiting for instantiation, fetching a waveform, and waiting for a response. The arcs are labeled with conditions that cause one to transition from one state to the next and with actions performed upon such a transition. This control structure may be implemented as a software object with a set of slots (the states) and attached methods. Methods test for state transitions and perform required actions. Recommendations for defining and simulating state machines are provided in the SDL Recommendation Z.100

E. Channel Agents

The characteristics of SPEAKeasy I and other software radios have been abstracted to the top-level software object diagram as shown below.
These high-level objects are called channel agents. Each software object has been allocated corresponding functions from the software radio functional block diagram. Those agents in the top row of the figure control the system, while those in the lower rows implement the traffic channels and related services. The objects of Figure implement the functions summarized in table given earlier. These radio functions are relatively generic. Thus, each waveform or mode of operation may be expected to have either its own set of agents or a set of parameters and a script that tell the agent how to treat that specific mode. Radio services, like bridging across waveforms, could be constructed as scripts or Java-like applets that interconnect channel agents and other high-level functions. The semantics of such top-level modules are readily represented in the unified modeling language (UML).

The timing and overall behavior of the channel agents are constrained by the lower level state machines that manage the channels. From a geometric perspective, agent subspaces in the [RAM] simplex subsume the {State machine} subspace. The extended instruction set for channel agents {Channel agents} is more of a category than a virtual instruction set. The top-level radio functions {Antenna and RF Control}, {Channel Control}, {IF Processor}, {Waveform Processor}, etc., each constitute a set of virtual instructions. Since these channel agents must be mutually consistent in order to work together, the class {Channel agents} refers to a mutually consistent set of such instruction set extensions comprising a radio mode.

**F. Distributed Layered Virtual Machine Reference Model**

The preceding sections have described essential software mechanisms employed in the construction of software radios. When viewed in terms of the topological spaces defined by the paths traced in the simplex of an equivalent RAM processor, layers of successively less machine-dependent software emerge. The distributed layered virtual machine is a way of representing the resulting hierarchy. Each layer is a virtual machine built on subordinate layers. The virtual machine interfaces may be held constant so that the implementation details and intellectual property present within a component at a given layer are hidden from all other layers. The software radio architecture that results from this process is illustrated in Figure given below.
As the capacity of FPGA’s and DSP’s continues to grow, the processing to support such virtual machines becomes more affordable. The services offered from the top-layer virtual machines thus are independent of the hardware implementations in the bottom-layer virtual machine. The FPGA and DSP hardware vendors or software tool suppliers, for example, could offer infrastructure layer software. This layer could be implemented in part using an augmented message-passing interface (MPI). The state machines in the infrastructure and radio applications layers would naturally fall into the domain of SDL. Third-party waveform vendors could offer middle layers. Interfaces among components may be represented in UML or the interface definition language (IDL), possibly including abstract syntax notation (ASN.1). Systems integrators and service providers could then define their unique added value in the top layers, building on the broad base of industry support at the lower layers. In addition, computer-aided software/systems engineering (CASE) vendors could embrace the wide range of reusable components to assist developers to encompass the entire software radio distributed virtual machine hierarchy for reduced time to market. The joint control function is distributed over these layers. Joint control at the top layer sets up channel objects and orchestrates services. In the radio applications layer, it consists of state machines that manage the radio channels. In the infrastructure layer, it establishes paths for signal and data flow. And in the hardware layer, it interacts with the operating system. Each layer accepts commands from...
higher layers, and returns status including fault conditions. With layering of the joint control function, the layered virtual machine insulates the higher layer applications from the details of the hardware.

G. Architecture Principle — Distributed Layered Virtual Machine Reference Model: Modules partitioned according to the distributed virtual machine reference model will insulate lower layer hardware-dependent modules from upper layer service-defining modules. The economic incentives for such a model center on integrating markets for economy of scale. The global wireless marketplace now consists of dozens of niches defined by unique hardware platforms and waveform-unique infrastructure. Industry is attempting to organize itself to integrate these diverse markets so that the next generation of wireless will offer low-cost plug-and-play services. The process of defining the required industry standards has already begun. The SDR Forum, for example, has defined an architecture framework based on functional threads in which a generic applications programmer interface (API) is being defined across the virtual machine. The forum is integrating the GloMo radio device API.

The object management group (OMG), on the other hand, has requested technology for CORBA real-time objects including multimedia at video data rates. A real-time CORBA might provide software radio infrastructure like an SDR API. The partitioning strategy and layered reference model presented here may provide insights helpful in evolving toward broadly accepted standards for future software-defined radios.