# A Policy Engine For Spectrum Sharing

Grit Denker, Daniel Elenius, Rukman Senanayake, Mark-Oliver Stehr, David Wilkins
SRI International
Menlo Park, CA 94025
{*firstname.lastname*}@sri.com

***Abstract*—We argue for a policy-based approach to increase spectrum availability. To this extend, we briefly summarize a new language for expressing policies that allow opportunistic spectrum access. A Policy Reasoner that reasons about these policies can be used with cognitive radios to guarantee policy-specified behaviors while allowing spectrum sharing. We present our policy reasoner design and we evaluated the reasoner in a demonstration. We describe the policies used in that demonstration and the results of the evaluation.**

## I. Introduction

Today, wireless communication is confronting two significant problems: spectrum scarcity and deployment delays. These problems derive from current procedures for the assignment of frequencies, which are centralized and static in nature [1]. The current scheme cannot adapt to the rapidly changing spectrum needs of users from the government, military, and commercial worlds. New technologies often cannot be used effectively because of this inflexibility, but they also provide the basis for solutions.

Spectrum is no longer sufficiently available, because it has been assigned to primary users that own the privileges to their assigned spectrum. However, studies [2] have shown that most of the spectrum is, in practice, unused most of the time. This observation was the starting point for DARPA's NeXt Generation (XG) Communications program, which proposes opportunistic spectrum use to increase spectrum availability. To achieve opportunistic spectrum use, radios must have the following capabilities

- Sensing over a wide frequency band and identifying primaries
- Characterizing available opportunities
- Communicating among devices to coordinate the use of identified opportunities Expressing and applying interference-limiting policies (among others)
- Enforcing behaviors consistent with applicable policies while using identified opportunities

Due to the large number of operating dimensions to be considered (frequencies, waveforms, power levels, and so forth) and the ever-changing nature of regulatory environments and application requirements, it is not feasible to design and implement optimal algorithms that allow radios to flexibly make use of available spectrum over time. Instead, a flexible mechanism has to be provided that supports spectrum sharing while ensuring that radios will adhere to regulatory policies. The solution must be able to adapt to changes in policies, applications, and radio technology. The XG Program has embraced a solution based on policies. The next section gives more detailed arguments for the policy-based approach.

We have implemented a device-independent Policy Reasoner (PR) that provides a software solution to opportunistic spectrum access. Our approach allows encoding of spectrum-sharing policies, ensures radio behavior that is compliant with policies, and allows policies to be dynamically changed. The PR either approves or disallows every transmission candidate proposed by a radio, based on compliance with currently active policies. Flexibility and spectrum sharing are achieved by expressing policies in a declarative language based on formal logic, and allowing devices to load and change policies at runtime.

Section II describes the advantages of using a policy-based solution. Our PR reasons with policies defined in our Cognitive (Policy) Radio Language (CoRaL), which was designed to support the definition of spectrum-access policies and to be extensible so that unanticipated policy types can be encoded. CoRaL has expressive constructs for numerical constraints and supports efficient reasoning. Section III gives a brief introduction to CoRaL that is sufficient for understanding the PR. In Section IV, we present various spectrum policies that illustrate key language features. Section V introduces the PR architecture and Section VI describes the design insights and implementation of the PR. We evaluated the PR in a demonstration, described in Section VII that uses the policies described in Section IV. We conclude with an overview of related work and plans for future extensions.

## II. Benefits Of Policy¬Based Spectrum Sharing

In current radios, policies are programmed or hard-wired into the radio and form an inseparable part of the radio's firmware. Typically, radio-engineers use imperative (procedural) languages such as C for radio software. One could envision implementing spectrum-sharing algorithms and behaviors on radios in a similar manner.

However, this approach has obvious drawbacks. Any change in policies requires reimplementation (and reaccredidation) in the firmware of every radio that might operate in bands affected by the changes. Clearly, this approach does not scale well as technological advances lead to an increasing number of radio designs. Further, it is not scaleable or flexible enough to deal with policies that are written by the many authorities in over 200 countries or that may initially change frequently as best practice is discovered or additional opportunities exploited.

The key difference in our approach is that declarative policies are expressed in terms of "what" should be protected or made available rather then "how" spectrum is protected or made available. Several considerations argue for this policy-based approach over encoding spectrum-sharing algorithms directly in radios:

- Radio behavior can quickly adapt to a changing situation. While policies themselves can be written to behave differently in different situations, the main advantage is that policies can be dynamically loaded without the need of recompiling any software on the radio. For example, a policy might be loaded to more aggressively exploit spectrum-sharing opportunities in emergencies.
- Policy changes can be limited to certain regions, frequencies, time-of-day or any other relevant parameter. Since policies are platform-independent, they can be loaded on different types of radios. Thus, new policies must only be uploaded into a radio to take effect, because each radio runs the policy reasoner on the currently loaded policies.
- Our approach decouples policy definition, loading, and enforcement from device-specific implementations and optimizations. One advantage is a reduced certification effort. When policies, policy reasoners, and devices can be accredited separately, accreditation becomes a simpler task for each component. Changes to a component can be certified without accrediting the entire system. We can certify the PR and each policy once, independent of the radio, and then test device configurations to see whether they correctly interpret PR outputs. (In effect, the cost of accrediting the policies and policy reasoner is shared across all radio platforms.) Radios can dynamically load accredited policies without additional certification.
- Another advantage of decoupling policies from radio implementation is that devices and policies can evolve independently over time. If a radio does not "understand" a policy, and can thus not fulfill its requirements, it will not have transmissions approved by this policy, thus missing opportunities but avoiding potentially creating interference. On the other hand, if a radio has more capabilities than required by a certain policy, it can just use what is required. Thus, new policies do not require changes in radio software or hardware, and existing policies will work on new radio hardware. Today a cyclic dependency exists where regulatory bodies must wait for technology and technology must wait to see what the policies look like.
- A policy-based approach is extensible with respect to the kinds of policies that can be expressed. While we already know many relevant parameters and the interrelationships between various categories of policies, including structural relations such as hierarchies, we cannot foresee the degrees of freedom in policy definition that may be desirable in the future. Our approach provides the means to define new policy parameters.

  Example parameters include functional allocations of spectrum (e.g., emergency response or aeronautical radio navigation), geographic restrictions (e.g., US vs. foreign policies), temporal restrictions (e.g., time-of-day), host nations or authorities (e.g., US vs. Europe, commercial vs. governmental), service restrictions (e.g., civil services, electronic warfare, Joint forces), and international vs. national policies on the same bands (e.g., maritime distress).
- An unprecedented amount of freedom and control of spectrum is possible as stakeholders can shape spectrum policies (as allowed by regulations) to best fit their objectives.

## III. CORAL POLICY LANGUAGE

The basis for policy-defined radios is a policy language that serves as an interface between at least two different viewpoints, namely that of the regulators and that of the radio engineers. For the sake of this discussion, we will assume each radio has a syste*m* strategy reasoner, which determines its strategy for making transmission requests by exploiting spectral opportunities. (Today's radios could be considered to have simple, hardwired strategies that do not exploit other opportunities.)

The main interest of regulators is the specification of admissible transmission behavior. They are usually not interested in how policy conformance is checked, as long as the check is correctly implemented. This is referred to as the soundness of the check. Regulators are not interested in the strategy used to discover opportunities, assuming that policy-conformance is ultimately enforced. Furthermore, they are not interested in verifying if a radio's strategy reasoner can exploit all transmission opportunities. Various trade offs (e.g., cost of sensing vs. need for spectrum), radio capabilities (e.g., ability to sense the spectrum), and the quality (degree of completeness) of the strategy reasoner itself will all affect which opportunities are exploited.

The main interest of radio engineers, on the other hand, is to exploit as many policy-conforming transmission opportunities as possible. This naturally leads to an incentive to enhance capabilities of both the strategy reasoner and the policy conformance reasoner.

To best support both of these viewpoints, a policy language with a simple and unambiguous semantics is needed. Since the foremost objective is to specify — as opposed to implement — policy-conforming behavior, a declarative language is a considerably better fit than an imperative language like C. In the XG project we have designed the Cognitive (Policy) Radio Language (CoRaL) [3], a domain-specific, logic-based specification language, which we briefly summarize.

## A. *CoRaL Concepts*

CoRaL is a typed fragment of first-order logic with equality, enriched by built-in and user-defined concepts [3]. Examples of domain concepts that are shared among most policies are: frequency, power, location, powermask, and signal.

A policy is composed of several *rules*. To support permissive as well as restrictive requirements, rules use either the `allow` or the `disallow` predicate, respectively. Policy rules are logical axioms that express under which *conditions* these predicates hold. These axioms can involve any declared parameters, which represent capabilities of the radio and the results of sensing actions (among other things).

Conditions can also use predicates, which express modes of operation, locations, and so forth. Thus, conditions allow for dynamic adjustment of policies to the current situation. For example, a rule could allow military radios to use the GSM band when a conflict starts, but not earlier. Clearly, such context-sensitive policies can respond to the situation in various ways, invoking either restrictive or permissive rules.

Numerical constraints are often used in policies specifications and can be directly expressed using built-in predicates in CoRaL. For example, a policy might require that for frequencies between 5000 and 5500 MHz, the transmission power should be at most 2dBm. A special syntax is available to specify powermasks, where the power is not constant but varies with frequency.

Restrictive (disallow) rules take precedence over permissive (allow) rules. A policy can also be extended by rules in another policy without causing logical inconsistencies. For example, one policy may have a rule allowing the use of frequencies 5000 to 5500 MHz, whereas another policy might disallow the use of frequency 5250 MHz, as well as allow frequencies between 5200 and 6000 MHz. Thus, the combination of policies will allow the use of frequencies between 5000 and 6000 MHz, with the exception of frequency 5250 MHz.

Policies and ontologies can refer to concepts defined in other ontologies with a *use* statement. This capability supports modular specification and reuse of policies and ontologies.

In addition to built-in types, variables, functions and predicates, CoRaL allows user-defined concepts. Formally, these concepts are expressed as equational or non-equational axioms in our logic. Concepts which are common across several policies can be factored out into ontologies, which can represent hierarchies of types and related functions or predicates. Formally, the only difference between ontologies and policies is that ontologies only define concepts and have no rules, whereas policies must have at least one rule and may also define concepts. Example ontologies that are useful for spectrum policies are discussed in the next section.

## B. *Ontologies*

In our work on XG, we defined the ontologies summarized in Figure 1. We have ontologies for basic types (such as bandwidth, frequency, and power), radio capabilities, evidence, signals, time, powermasks, transmissions, and request parameters (among others). These ontologies give an extensible base for the parameters over which policies can be formulated,

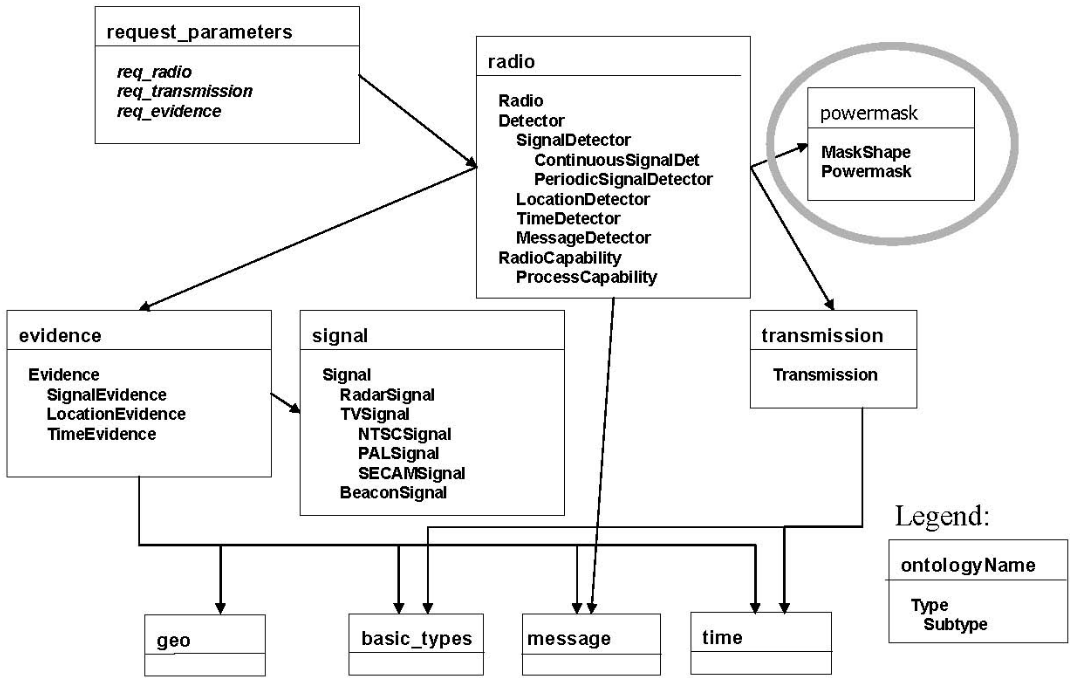


Figure 1. Example Ontologies. Each box represents an ontology. The name on the top of the box is the name of the ontology. An arrow between two ontologies indicates that the ontology at the tail of the arrow uses the ontology at the head. For brevity, we only list types (in arial font) and variables (in italic font) in some of the ontologies. Functions, predicates, and axioms are not shown. Type hierarchies are represented using indentation.

but should not be regarded as the only possibility. CoRaL expresses ontologies and domain concepts using type and subtype declarations. New requirements can be captured in user-defined ontologies, which may also build on these basic types.

Our request-parameter ontology defines three variables that are typically contained in a transmission request. The variables are *req radio : Radio*, which describes characteristics of the requesting radio; *req transmission : Transmission*, which details parameters of the requested transmission, such as frequency and power; and *req evidence : Evidence*, which contains one or more evidence objects, each of which generally pertains to location, signal, or time of sensed data that was collected by the radio.

These parameters refer to concepts such as *Transmission, Radio*, and *Evidence*. These concepts are modeled in CoRaL as types distributed over various ontologies, and build on the basic types shown in 1. Typically, a concept defines several operations.

As an example, we show more detail about the ontology for transmission. One operation on the *Transmission* type is a function that determines the center frequency of the requested transmission:

*centerFrequency: Transmission* $\rightarrow$ *Frequency*

Other operations, such as mean EIRP (Effective Isotropic Radiated Power), are defined in a similar fashion. The formalization of the transmission ontology in CoRaL is given below.

```
ontology transmission is
  use time,basic_types;

  public type Transmission;

  public const  centerFrequency :
                Transmission -> Frequency;
  public const  bandwidth :
                Transmission -> Bandwidth;
  public const  maxOnTime :
                Transmission -> TimeDuration;
  public const  minOffTime :
                Transmission -> TimeDuration;
  public const  meanEIRP :
                Transmission -> Power;
  public const  transmittedBy :
                Transmission -> Transmitter;
end
```

Powermasks are another typical concept in spectrum policies. As an example, the powermask for DFS (Dynamic Frequency Selection) [4] is depicted in Figure 2 and formally defined below using CoRaL syntax.

We provide an intuitive syntax for defining powermasks, most of which are either linear or step functions. CoRaL represents a powermask as a list of tuples of numbers *(x,y)*, where *x* refers to the frequency values on the x-axis (in MHz) and *y* refers to the power values on the y-axis (in dBc). CoRaL connects the points either as a linear or step function, as indicated by a keyword preceding the list of tuples. For example, the DFS powermask in Figure 2 is represented in

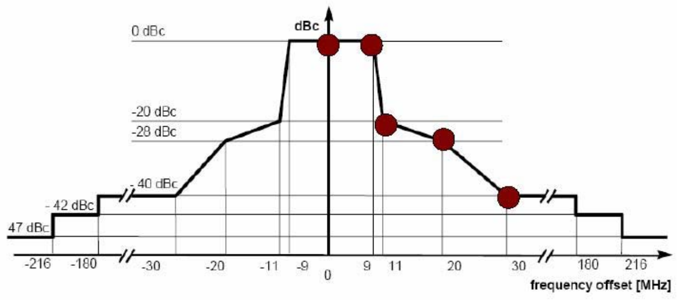


Figure 2. Example DFS Powermask. This figure (from [4]) has been annotated to highlight points in the graph corresponding to tuples in the CoRaL representation of the powermask.

CoRaL as follows. (The five circles in the figure correspond to the first five tuples.)

```
defconst maxInBandLeakage : Powermask =
  symmetric linear
  [(0, 0), (9, 0), (11, 20),
   (20, 28), (30, 40), (180, 40),
   (180, 42), (216, 42),
   (216, 47), (inf, 47)]
```

## IV. Spectrum Policy Examples

For the experiments described in Section VII, we wrote listen-before-talk policies [5] over a broad range of frequencies. The policies were chosen to illustrate some of the main features of CoRaL (but they do not exhibit all language features), and to be realistic and thus potentially relevant for spectrum-sharing radio operations in the field.

Listen-before-talk policies require a radio to actively sense its environment and submit data to the reasoner about what other signals were detected at what power levels. To add realism, we also included operational phases and geographical information as parameters.

We defined example operational phases, such as “Day-to-Day”, “Natural Disaster”, and “Training and Testing”, each with policies that address the specific communication needs of that phase. To mimic two countries with different regulations, we defined two adjacent regions that would be traversed by the radios. The radio must submit information about location and phase to the PR to get transmission requests approved by policies that require those parameters. The PR will apply the policies appropriate to the location and phase of a request.

Figure 3 depicts three (of many possible) categories of policies used in our experiments. While our work has not formalized any such higher-level classifications, these could provide utility and aid understanding. For example, the categories in Figure 3 could correspond to increasingly higher-level regulatory agencies.

The policies in the inner circles generally require increasingly more information in transmission requests. Thus, a policy belonging to the innermost circle often requires information about operational phases, location, frequencies, and sensed state of the spectrum. The policies in the middle circle only check for operational phases, frequencies and state of the spectrum, while policies in the outermost circle might only constrain the frequencies to be used.

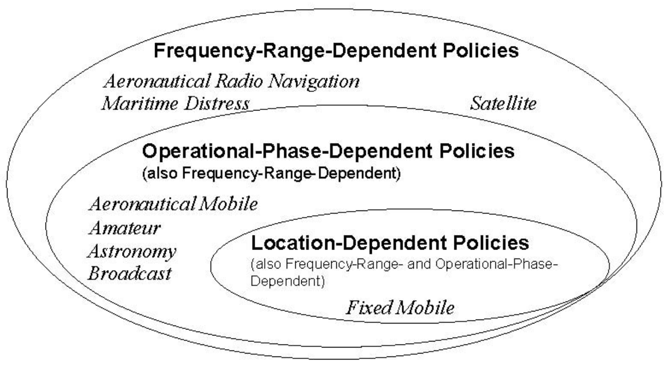

Frequency-Range-Dependent Policies
Aeronautical Radio Navigation
Maritime Distress
Satellite
Operational-Phase-Dependent Policies
(also Frequency-Range-Dependent)
Aeronautical Mobile
Amateur
Astronomy
Broadcast
Location-Dependent Policies
(also Frequency-Range- and Operational-Phase-Dependent)
Fixed Mobile

Figure 3. Classification of encoded policies used in our experiments.

Some of the policies are *permissive* and allow use of spectrum under given conditions. Other policies are *restrictive* and forbid or restrict the use of bands. Figure 4 summarizes some of the policies used in our experiments. The restrictive policies in row 2 forbid access to the Satellite, Aeronautical Radionavigation, and Maritime Distress bands, as indicated by the *Protected* keyword. (We show details of such a policy below).

| Threshold in MHz for Band and Operational Phase in Region 1 (Different Threshold for Region 2, if applicable) | Day-To-Day | Special Event | Natural Disaster | Testing and Training |
|---|---|---|---|---|
| Satellite<br>Aeraonautial Radionavigation<br>Maritime Distress | Protected | Protected | Protected | Protected |
| Amateur | No Policy | -100 | -80 | No Policy |
| Astronomy | No Policy | -100 | -80 | No Policy |
| Broadcast | No Policy | -90 | -80 | No Policy |
| Aeronautial Mobile | No Policy | No Policy | -100 | No Policy |
| Fixed Mobile 1 | -100 (-90) | -100 (-90) | -80 | -100 (-90) |
| Fixed Mobile 2 | No Policy | -100 (-80) | -100 (-80) | No Policy |

Figure 4. Overview of encoded policies.

For other bands, permissive policies define the strength of signals that can be sensed by the radios and still allow transmission. The thresholds depend on the operational phase. For example, for the broadcast bands, a policy states that if the radio is operating in the "special event" phase and only senses signals that measure 90 dBm or less, then transmission is allowed (we show details of this policy below). If the radio was operating during a natural disaster, it would be okay to transmit even in the presence of signal up to 85 dBm. For some policies, different thresholds are given for the same frequency band in the two regions to illustrate how different regulatory bodies (different nations, service providers, and so on) might allow the use of spectrum under different circumstances.

Some of the of frequency bands we used for the Fixed Mobile policies (all in MHz): 3074.8, 75.287.2, 225328.6, 335-400, 420450, 12401390, 17551850, and so on. These frequency ranges reflect the current assignments of these bands for non-federal government in the U.S. [6]. However, our goal was not to capture current practices and policies. Our focus was on evaluating CoRaL and its reasoner. To test the expressiveness and utility of our language, we have successfully implemented in CoRaL major parts of the Dynamic Frequency Selection (DFS) algorithms for the unlicensed 5 GHz band [4].

Part of the CoRaL encoding of the restrictive policy for the aeronautical radionavigation band follows.

```
policy aeronautical is
 use request_params;
 use mode;

 disallow if
   (mode(Day-to-Day) or mode(SpecialEvent)
   and
   (centerFrequency(req_transmission)
                in {74.8 .. 75.2} or
    centerFrequency(req_transmission)
                in {108.0 .. 117.975} or
    ...
    centerFrequency(req_transmission)
                in {2700.0 .. 2900.0});

end
```

The policy imports the `request_param` ontology (and the ontologies imported by this ontology via transitivity of import), which defines concepts such as `centerFrequency`, `req_transmission`, and the modes. This policy has only one rule, which disallows the use of any of the specified frequency ranges in the specified modes.

The following policy combines phase, sensed evidence about the spectrum state, and location. The first `allow` rule makes the specified frequency ranges available for transmission, for radios that are in `Day-to-Day` or `TestingAndTraining` mode and have sensed signals of less than 115 dBm. The second rule allows access to another set of frequencies during a `SpecialEvent`, but only if the radio is located in region `r1`. Whereas the location information restricts use, the threshold of the second rule is more permissive (95 dBm instead of 115 dBm).

```
policy fixedMobile is
  use request_params;
  use mode;
  use region;

  allow if
   (centerFrequency(req_transmission)
                    in {225.0 .. 328.6} or
    ...
    centerFrequency(req_transmission)
                    in {2200.0 .. 2290.0})
   and
   (mode(Day-to-Day) or mode(TrainingAndTesting))
   and
   ((exists ?se:SignalEvidence)
        req_evidence(?se) and
        peakRxPower(?se) =< 115.0);

  allow if
    (centerFrequency(req_transmission)
                    in {30.0 .. 74.8} or
     ...
     centerFrequency(req_transmission)
                    in {2900.0 .. 3000.0})
    and
    (mode(SpecialEvent)
     or
     ((exists ?le : LocationEvidence, ?l : Location)
        req_evidence(?le) and
        location(?le) = ?l and
        locationInEllipse(?l,r1) = true))
  and
    ((exists ?se:SignalEvidence)
       req_evidence(?se) and
       peakRxPower(?se) =< 95.0);
```

## V. PolicyBased Radio Architecture

The XG architecture [7] (see Figure 5) consists of the radio hardware and firmware, which includes the RF frontend as well as sensors; the System Strategy Reasoner (SSR), which is typically specific to the radio hardware and can perform low-level tuning and real-time optimizations; and the platform-independent Policy Reasoner (PR), which determines whether transmission requests from the SSR conform to the currently loaded policies. The SSR must not transmit unless it has received message from the PR that the transmission is allowed.

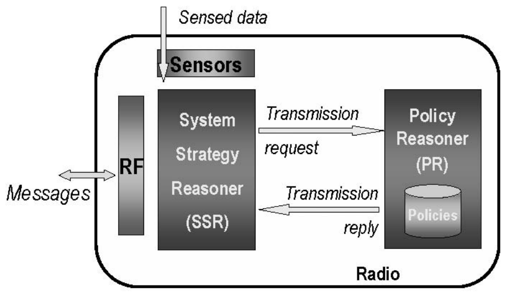


Figure 5. XG Architecture. The small boxes are hardware components. The SSR is a radio component that exploits transmission opportunities. This is one of many possible architectures for using our policy language.

There are several different types of messages:

- *RF-SSR*. All incoming messages to the XG radio arrive at the RF unit, and end up in the SSR.

  These messages can be control messages, such as updates to system strategies, updates to policies, or messages controlling the coordination with other radios. Similarly, all messages going out from the XG radio originate in the SSR and are passed through the RF component. Outgoing messages can also be control messages (acknowledgment of policy updates, requests for new control channels, etc.) or data messages.

- *Sensors-SSR*. The details of this interface will be determined by the radio designer. We assume that the sensors send their received data (or conclusions drawn from it) to the SSR. The analysis of sensor data, sensor data aggregation, signal detection, and other such processing could happen in the sensor component(s), in the SSR, or in a dedicated component (not shown). The SSR may send control messages to the sensor components.

- *SSR-PR*. There are several types of messages in the interface between the SSR and the PR. **Transmission requests:** Before an XG radio can send a transmission, it needs approval from the PR. The SSR builds a transmission request, and sends it to the PR. The PR reasons about the request and the active policies, and responds by sending one of three replies to the SSR: (1) The transmission is allowed. (2) The transmission is not allowed. (3) The PR returns constraints that must be satisfied. Given acceptable values of the underspecified request parameters, the transmission will be allowed. **Policy updates:** The SSR can also send policy-update messages to the PR, in order to add or remove policies to and from the PRs policy base and to activate or deactivate policies. **Policy information:** The SSR can request information regarding which policies are loaded or active.

## VI. PrologBased Policy Engine

SRI International (SRI) has been mainly concerned with the Policy Reasoner (PR) module of the XG architecture. We have implemented a prototype PR in Prolog[1], which gives yes/no answers to transmission requests, covering a large subset of the CoRaL language. We are currently investigating options for implementing a PR with the ability to return constraints on failed requests.

### *A. Policy Reasoner Components*

Figure 6 shows an overview of the main modules of the PR, and some of the relationships between them. This architecture is not part of the XG Architecture. The PR could have been implemented differently. The larger ovals show the implementation languages. Most of the modules are implemented in Java, but the reasoning is done in Prolog. The SSR can be implemented in any language.

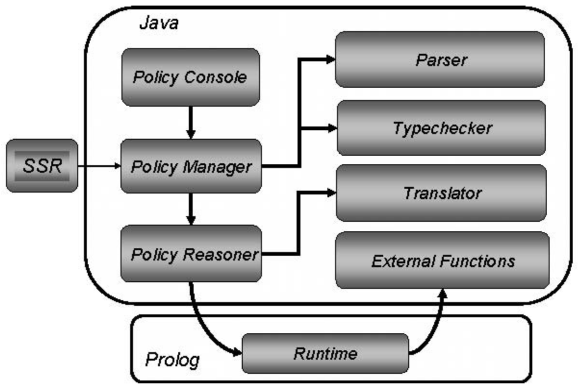


Figure 6. Policy Reasoning Engine Modules

From the perspective of the SSR, the PR can be viewed as a black box, with a certain input/output behavior defined by the SSRPR interface described in Section V.

**SSR**. All messages to the PR originate from and are initiated by the SSR. The PR never initiates communication itself.

**Policy Console**. A text-based interactive application that can be used to debug and test policies, perform requests, etc.

[1] The reasoner can be provided to interested parties upon request, subject to approval from our sponsor DARPA.

**Policy Manager.** This module controls all operations of the PR. All of the methods in the SSRPR interface are implemented as static methods of the PolicyManager Java class. The Policy Manager delegates functionality to various other modules, as can be seen in Figure 6. This module keeps lists of all currently loaded/active policies.

**Policy Reasoner.** This module is a thin Java wrapper around the module that does the actual reasoning. There is a Reasoner Java interface with only a few methods, and one implementation, PrologReasoner. This means that we could replace the reasoner backend without changing anything else in the implementation. The main functionality in the PrologReasoner class is

- Translating policies, ontologies, and requests from abstract syntax to Prolog.
- Loading and unloading these modules to/from Prolog.
- Asking Prolog to perform the actual queries, and returning the results as a Java object (of the Result class).

**Parser**. This module takes a CoRaL policy, ontology, or request as input and produces abstract syntax, in the form of Java objects, as output. The Java code for the parser is generated automatically at build time from a grammar specification, using the JavaCC parser generator.

**Typechecker**. Type checking is done on the abstract syntax, i.e., on Java objects. After being type checked, the abstract syntax is supplemented with some type information, which is used in the translation to Prolog, among other things.

**Translator**. This module translates policies, ontologies, and requests from abstract syntax to Prolog (see the following section).

**External Functions**. Some of the functionality needed by CoRaL policies cannot (or should not, due to efficiency concerns) be implemented in CoRaL itself. These functions have been implemented in Java. So far, the external functions fall into the following categories:

- Powermask operations
- Time operations
- Geometric operations

**Prolog Runtime**. This module is the only part that is implemented in Prolog. It consists of

- The translated policies, ontologies, and requests.
- Code for equality and function evaluation, which is not normally supported in Prolog, and therefore has to be encoded (see the next section).
- Code for the built-in predicates and functions (<, +, mod, etc).
- Code for marshalling parameters between Prolog and the external functions in Java.

## B. *CoRaLtoProlog Translation*

When implementing a language (the object language) using another language (the meta-language), there are two main approaches one can use:

- The object language is translated to the meta-language.
- The object language is encoded and evaluated in the meta-language.

The first approach is generally more efficient, because there is no overhead involved in evaluation. The object language is essentially executed by executing the meta-language. However, one is limited to the operational behavior of the meta-language. With the second approach (encoding) one has more flexibility, since one can implement the desired operational behavior in the meta-language.

Mixes of the two approaches are also possible. The PR described in this report uses such a hybrid approach. The following parts of the language are translated: CoRaL predicates are translated to Prolog predicates, CoRaL "standard" rules become Prolog rules, CoRaL policies, ontologies, and requests are translated into Prolog modules. There were some differences between how Prolog handles modules and CoRaL handles them, so this was not a completely straightforward translation. Finally, CoRaL abstract data types become Prolog unary predicates.

However, Prolog's native operational semantics does not support equality constraints, user-defined equations or function evaluation. The only way to achieve those features in Prolog is to use the encoding approach. The encoding is rather shallow:

- Equality constraints are represented by a Prolog predicate eq/2, defined in the pre.pl file. There are Prolog rules for the transitivity and reflexivity of equality.
- User-defined equations are represented by a rewrite/2 predicate, and are always interpreted in a directional left to right. These equations are taken into account when equality constraints are evaluated.
- Functions are represented as compound Prolog terms. They are evaluated, when the arguments are provided, by applying rewrite rules, in a call-by-value (eager) evaluation fashion.

## C. *Example Translation: CoRaL to Prolog*

We present an example of the translation (or encoding) of a CoRaL policy to Prolog. We use a listen-before-talk example policy that uses time and location information. Such information is often used in policies relating to television broadcast.

The policy allows XG radios to transmit in the band 450 MHz to 600 MHz if they (1) transmit continuously for at most 1 second with (2) a bandwidth of at most 6 MHz, (3) have off-time of at least 150 milliseconds, (4) have duty cycle of at least 50% across a 2second period, (5) if they sample spectrum at least once every 5 seconds for TV signals using power sensing but no subnoise detection of DTV signals, (6) they received spectral power in the channel they are transmitting equivalent to or less than 100 dBm, and (7) the peak power spectral density of their emission does not exceed 53 dBm/Hz. This policy is encoded in CoRaL as follows:

```
policy tv1 is
  use ssc_params;
  defconst F : Frequency =
      centerFrequency(req_transmission);
  allow if
    F in {450.0 .. 600.0} and
    timeDurationLessThanOrEqual(
         maxOnTime(req_transmission),
         td(0,0,0,1,0)) = true and
    bandwidth(req_transmission) =< 6.0 and
    timeDurationLongerThanOrEqual(
         minOffTime(req_transmission),
                    td(0,0,0,0,150)) = true and
    meanEIRP(req_transmission) =< 53.0 and
    (exists ?se:SignalEvidence)
      req_evidence(?se) and
      peakRxPower(?se) =< 100.0 and
      {F3.0 .. F+3.0} in
           scannedFrequencies(?se) and
      (exists ?d:PeriodicSignalDetector)
        detectedBy(?se) = ?d and
        sampleRate(?d) >= 0.2 and
        dutyCycle(?d) >= 0.5;
end
```

The encoding is

```
Header
:module(tv1,[]).
:style_check(singleton).
:style_check(discontiguous).
:dynamic allow_l/1.
user:allow(X) :
  context_module(M),
  debug:debug(runtime,debug,"~n~w:allow",[M]),
                 allow_l(X).
rewrite_l(if(F,M1,M2),X) :
        call(F) > X=M1 ; X=M2.

%Rules
allow_l(tv1) :
  in(centerFrequency(req_transmission),
    range(450.0,600.0)),
  eq(timeDurationLessThanOrEqual(
       maxOnTime(req_transmission),
               td(0,0,0,1,0)),true),
  lte(bandwidth(req_transmission),6.0),
  eq(timeDurationLongerThanOrEqual(
       minOffTime(req_transmission),
               td(0,0,0,0,150)),true),
  lte(meanEIRP(req_transmission),
       uminus(53.0)),
  ('SignalEvidence'(Se),
    req_evidence(Se),
    lte(peakRxPower(Se),100.0),
    in(range(sub(centerFrequency(req_transmission),
                3.0),
       add(centerFrequency(req_transmission),
          3.0)),
   scannedFrequencies(Se)),
  ('PeriodicSignalDetector'(D),
   eq(detectedBy(Se),D),
   gte(sampleRate(D),0.2),
   gte(dutyCycle(D),0.5))).
```

In these policies, we can observe the shallow encoding of function applications as compound Prolog terms, the representation of built-in predicates (e.g., eq for equality, lte for less than or equal), and so forth. We also see that the defined constant F has been substituted, because Prolog does not natively support definitions. So `centerFrequency (req_transmission)` occurs everywhere in place of the defined constant `F` (which has the same behavior). We can also see how some special syntactic forms of CoRaL have been translated, for example, `{450.0 .. 600.0}` became `range(450.0,600.0)`.

We could have accomplished this in another programming language. However, we make use of goal-directed reasoning, which is well supported by Prolog. We mentioned that abstract data types are translated to unary Prolog predicates. We can see two examples above: `'SignalEvidence'` and `'PeriodicSignalDetector'` (these are in single quotes because Prolog would otherwise interpret them as variables). Let us look at the existential quantifiers in the policy above. We have `(exists ?se:SignalEvidence)....`

In the Prolog translation, this becomes `'SignalEvidence' (Se)`.

?se has become Se because variables have to start with an uppercase letter in Prolog. This translation has the correct behavior, because of Prolog's *unification* and *backtracking* features. Prolog will see `'SignalEvidence'(Se)` as a goal that must be satisfied. This will happen if there is a rule that unifies with this goal. For example, if we have, in a request, a line `sigev : SignalEvidence;` this is translated to the Prolog version `SignalEvidence(sigev)`

This will unify with `SignalEvidence(Se)`, with the obvious substitution Se = sigev. Prolog can now continue to the next goal, keeping this substitution for whenever the Se variable next appears. If some further goal fails, Prolog will backtrack to the same goal again, and try another value for Se, if there is one. Therefore, the existentially quantified formula will succeed if and only if the Prolog version succeeds. Of course, this kind of execution can be very slow if there are many unifiers, and only a few of them succeed.

## *D. Limitations of Prolog Encoding*

The Prolog encoding of policies works essentially by *executing* them. This is usually very efficient, but is also has serious drawbacks.

*Universal quantifiers.* We cannot translate all kinds of universal quantification to native Prolog. Some kinds are not problematic. For example, the universal quantifiers on the outside of rules can be handled because they are already implicitly in Prolog rules. Also, some kinds of finite universal quantifiers are handled by trying all possibilities. For example, `(forall x:Int in [1,2,3]) p(x)` is translated to `forall(X,(member(X,[1,2,3]), p(X)))`. Prologs forall operator does what we just said: It tries all possibilities, i.e. all unifiers. So, in this case, X will be unified first with 1, and the p(1) goal will be tried. If this succeeds, X is unified with 2, and p(2) will be tried, and then the same for p(3). Only if all three goals succeed does the whole forall statement succeeds. Our comments on the efficiency of using unification for existential quantification are true for universal quantification and result in even worse efficiency. In the existential case, we can at least quit when we find a unifier that makes the existential statement succeed. In the universal case, we always have to try all substitutions.

*Constraints.* In the future, we plan to extend the PR to return those constraints that still have to be satisfied before an underspecified request can be approved. These constraints can guide the SSR towards transmission opportunities under the current policies.

The PR must return constraints when a policy fails due to the request being underspecified. For example, given a request for certain frequencies, the PR might return a constraint on the maximum power. While Prolog does have the so-called clp/r constraint-solving capability for linear arithmetic, we have found this to be inadequate because of lack of support for negations and quantifiers.

Given these limitations and the fact that only a fragment of CoRaL has been implemented currently, we nevertheless have been able to encode and execute all the policies that were desired for our XG experiments and demonstration.

## VII. Reasoner Capability Demonstration

XG technology was demonstrated for the first time to key stakeholders at Fort A.P. Hill in Virginia, in August 2006. The demonstration included the functionality of XG Radios developed by Shared Spectrum Company and the CoRaL language and the Policy Reasoner developed by SRI International. The test was carried out in the geographic region (known as the 'Drop zone') shown in the map on the upper left hand corner of Figure 7 (Marked as [A]).

Pairs of XG radios formed communication links and traversed a path from the region shown as 'Metropolitan Area' to the 'Disaster Area' and back. Throughout the drop zone, legacy radio pairs were placed and formed a separate set of communication links. The legacy radios were used for the purpose of generating interference in the communication channels used by the XG radios and to test if such interference is detected by the XG radios and, if so, whether the channel would be abandoned and the XG radios would switch to a channel which would not cause interference.

The tests conducted at the Fort A.P. Hill drop zone successfully demonstrated that the XG Radios were adapt at detecting interference and changing communication channels rapidly to avoid interference. In fact, the channel switch was so fast that the legacy radios were not affected, showing readiness for real-life scenarios.

The XG radios in this demonstration did not use SRI's PR, but instead used a simpler reasoner with a less powerful policy language, but one that had the same form/fit/function as SRI's PR. The standalone demonstration of SRI's PR demonstrated the wide array of policies that the CoRaL language can express, and the speed and scalability of the PR, which was fed thousands of transmissions requests in a few seconds. All the policies summarized in Figure 3 of Section IV were active.

To test the scalability of the PR, we generated requests for transmission for the entire scan range of the Rockwell sensor used by the XG radio (from 20MHz to 2500MHz) in 1MHz increments. Therefore, for each pass of the sensor, the PR received approximately 2500 requests, which is many more than any realistic SSR would send. The PR processed about 200 requests per second. This 5ms average time is more than adequate to support the rapid abandonment time required by XG to avoid interference.

The graphical user interface of the PR demonstration consisted of two windows (Figure 7 for inputs and Figure 8 for outputs). The four parameters that make up a request are

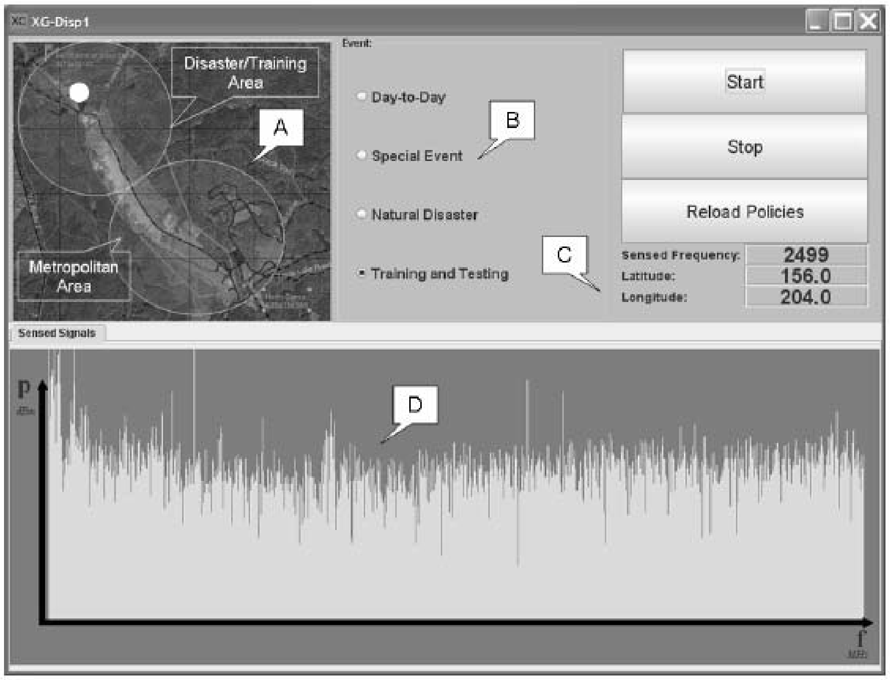
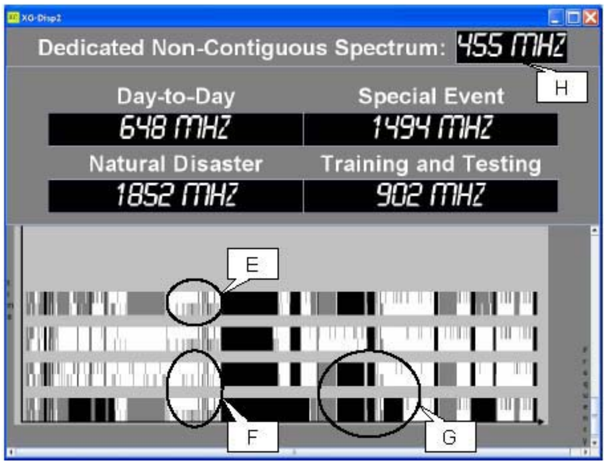


Figure 7. Request Parameters GUI.


XG-Disp2

Dedicated Non-Contiguous Spectrum: 455 MHZ

| Day-to-Day | Special Event |
|---|---|
| 648 MHZ | 1494 MHZ |
| **Natural Disaster** | **Training and Testing** |
| 1852 MHZ | 902 MHZ |

Figure 8. Reasoner Answers GUI.

location, operational phase, frequency and sensed signal strength, which are depicted by regions marked AD respectively in Figure 7. The drop zone marked [A] consists of two sub-regions "Metropolitan Area" and "Disaster Area", which were used to depict the PRs ability to do geographic reasoning using latitude/longitude coordinates. The underlying policies change as the radio moves between regions or changes operational phases.

Transmission requests are defined in CoRaL. A typical request in the PR demonstration had the following form:

```
request xgreq is
  centerFrequency(req_transmission) = 1250.00;

  public const se : SignalEvidence;
  req_evidence(se);
  peakRxPower(se) = 79.0;

  mode(SpecialEvent);

  public const le : LocationEvidence;
  req_evidence(le);
  location(le) = (112.0, 215.0);
end
```

Figure 8 depicts the results returned by the PR for the set of requests submitted to it. The field marked [H] in this figure denotes the total spectrum currently available for military use in the 20MHz2500MHz range (see [6]).

Using the policies summarized in Section IV, we demonstrated the use of CoRaL policies to dynamically change how aggressive the radio is in accessing spectrum, based on the location of the XG radio, its operational mode, and the sensed signal strengths.

The demonstration was carried out by changing the operational phase in the following sequence: Day-to-Day, Special Event, Natural Disaster, Training and Testing. For each phase, we ran the 2500 requests twice with different sensed data. The results of these requests are illustrated on the bottom of Figure 8 using color coding for the reasoner answers. Each black/grey/white stripe corresponds to the answers for an operational phase, starting with the Day-to-Day phase on the bottom. On the top of Figure 8 are four fields that show the total available spectrum for each of these operational modes using an XG radio. As expected this amount increases noticeably from the first operational mode to the third monotonically and is consistent with the behavior specified in the CoRaL policies. The total available spectrum for the operational phase "Special Event" is an intermediate value and depicts the ability of the PR to be fine tuned to fit custom needs.

Three colors, White, Black and Gray are used in Figure 8 to denote different responses from the XGPR for a request. We used the following color coding for answers from the PR (see Figure 9). Black denotes a frequency band which is explicitly protected by one of the active policies. White denotes a band for which transmission is allowed, either because it is assigned or because there is a sharing opportunity given current policies and request parameters. Gray denotes a band for which transmission was disallowed. Given the active policies in the demonstration, this generally indicates an unacceptable probability of causing interference.

| |
|---|
| **Available Spectrum** |
| **Currently Unavailable Spectrum** |
| **Unavailable Spectrum (Protected Band)** |

Figure 9. Color Coding of Reasoner Replies in GUI.

The region marked [E] depicts the main characteristic of the XG technology. In this case, none of the input parameters has changed except for the location of the XG radio and the strength of the sensed signals. A closer examination shows that in this situation certain requests that were denied are now approved, due to the change in location or sensed signals. Such opportunistic spectrum access is the key achievement of the XG technology and shows the policy-based radios adopting to dynamic situations rapidly (the change from Gray to White and vice versa).

Regions marked [F], [G] shows that the PR can change its behavior to more aggressive access of spectrum by either removing protections from protected frequency bands [G] (black turns white or gray) or by increasing its thresholds for interference [F] (grey turns white). The latter will, of course, increase the probability of interference, but, at any given time, the PR ensures that the radio behaves according to the currently loaded policies, whatever those might be.

Regions denoted in white in Figure 8 depict requests that were approved. These represent opportunistic spectrum access that conforms to active policies, which implies an acceptably low risk of interference.

## VIII. CONCLUSION

Current radio technology allows us to aggressively access spectrum by ignoring sensed signals or preset thresholds, but the risk of interference is high. The policy-based XG technology provides a framework within which dynamic behaviors for radios can be specified by policies so that spectrum can be accessed opportunistically with an acceptably low risk of interference.

CoRaL has proven expressive enough to encode a wide variety of spectrum-sharing policies, including DFS and all the policies that were desired for our XG experiments and demonstration. In our experiments, the PR processed requests in less than 5ms on average, which is more than adequate to support the rapid abandonment time required by XG to avoid interference.

**Future Work.** Due to the expected complexity of future policies, a policy language should allow for advanced forms of policy analysis, e.g. detection of logical inconsistencies. We are currently investigating the use of theorem proving technology to provide specialized analysis methods for spectrum policies written in CoRaL.

The functionality of the PR will also be extended. Our current PR does only validate transmission requests and provides yes/no answers. However, smart radios could exploit the reason for which a transmission request failed, if that information were available. Our next generation PR will provide more detailed answers in the case of negative decisions. In particular, we will return the additional constraints, if any exist, that the radio must satisfy in order to be granted use of the spectrum.

Finally, we will perform more experiments with larger policy sets or more complex policies to test the limitations of the reasoner.

## ACKNOWLEDGMENT

This research was supported by DARPA's neXt Generation (XG) Communications Program under Contract Numbers FA875005C0230 and FA875005C0150.

## REFERENCES

[1] J. A. Stine and D. L. Portigal, "Spectrum 101. An Introduction to Spectrum Management," MITRE, Technical Report MTR 04W0000048, 2004.

[2] W. J. Byrnes and M. McHenry, "In the matter of Establishment of an Interference Temperature Metric to Quantify and Manage Interference and to Expand Available Unlicensed Operation in Certain Fixed, Mobile and Satellite Frequency Bands," FCC Comment, ET Docket No. 03237, Apr. 2004.

[3] G. Denker, D. Elenius, R. Senanayake, M.O. Stehr, C. Talcott, and D. Wilkins, "XG Policy Language. Request for comments." SRI International, Tech. Rep., 2006.

[4] "ETSI Standard EN 301 893 V1.2.2 (200306)," 2003, reference DEN/BRAN0020002. [Online]. Available: http://www.etsi.org

[5] A. E. Leu, M. McHenry, and B. L. Mark, "Modeling and analysis of interference in ListenBeforeTalk spectrum access schemes," International Journal of Network Management, vol. 16, no. 2, pp. 131–147, 2006.

[6] "United states frequency allocations for radio spectrum," October 2003.

[7] D. Elenius, G. Denker, and D. Wilkins, "XG Architecture. Request for comments." SRI International, Tech. Rep., 2006.