CoRaL - Policy Language and Reasoning Techniques for Spectrum Policies

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

***Abstract*— We present Cognitive Radio (Policy) Language (CoRaL), a new language for expressing policies that govern the behavior of cognitive radios that opportunistically share spectrum. A Policy Reasoner validates radio transmissions to ensure that they are compliant with the spectrum policies. The Policy Reasoner also discovers spectrum sharing opportunities by deriving what requirements have to be fulfilled for transmissions to be valid, i.e., in compliance with policies. A novel mix of reasoning techniques is required to implement such a reasoner.**

I. Introduction

Because of the centralized, static nature of current spectrum allotment policy, wireless communication is confronting two significant problems: spectrum scarcity and deployment delays.

With increasing demands on wireless communications, spectrum is becoming a hot commodity, with national spectrum regulators auctioning out frequencies for incredible prices[1].

Deploying radios to different regions or countries means encountering a new spectrum environment, with new policies to abide by. Delays occur because policies are currently hard-coded into the radios' software or firmware, and time-consuming upgrades must be performed.

The solution to these problems offered by DARPA's NeXt Generation (XG) Communications Program is based on two key observations: a) In a typical situation, most of the spectrum, although allocated, is not in use; and b) Radios could be made more flexible by adopting a scheme of declarative spectrum policies, offering the usual advantages of a policy-based approach, such as easier deployment, verification, management, etc.

Under the new approach to spectrum management, highly capable sensors are used at runtime to scan allocated frequency bands in order to detect parts of the spectrum that are not currently in use. Policies are used to describe the constraints on using the spectrum. For example, how long must a band be empty before it can be used, what level of detected spectral density is considered background noise, and at what power level are you allowed to transmit? These policies vary depending on geographic region and time, and thus radios must be able to load new policies at runtime.

The overall picture is that each XG radio is equipped with a Policy Reasoner (PR). The radio provides the PR with facts about itself and the (sensed) environment, and the reasoner tells the radio whether it can transmit or not.

SRI was tasked with developing a policy language and reasoning techniques for the XG program.

II. Requirements

There were several requirements on the policy language and reasoner.

Accreditability. It should be possible to accredit the policy reasoner, individual policies, and radio software, all independently of each other. This reduces the combinatorial nightmare of having to accredit each combination of radio and policies, as is done today. Furthermore, it removes the need to re-accredit the radio and its software every time a new policy is introduced, which currently makes the goal of rapid deployment impossible.

Extensibility. It should be possible to easily add new domain knowledge and policies, as well as to extend the expressiveness of the language if needed for new domains. To achieve the latter goal, we based the language on a solid and well-understood logical foundation.

Expressiveness. We examined current spectrum policies, as well as imagined future ones, in order to determine what kind of constraints we need to be able to express in the language. In particular, we need the following features:

[1]see e.g., `http://wireless.fcc.gov/auctions/`

- *Functions*, such as the powermask in Figure 1 from the DFS policy [1].
- *Computations*, such as temporal and geospatial reasoning.
- *Orderings* (e.g. frequency less than 5000.0 MHz).

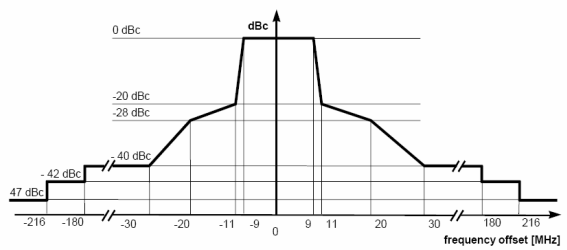

Fig. 1. Powermask from Dynamic Frequency Selection (DFS) policy.

Opportunity Discovery. We need the PR to not only say yes or no to transmission requests from the radio, but to also, where applicable, provide an answer of the form "yes, if C", where some additional constraints C need to be satisfied for the transmission to be approved. This is required because the radio cannot simply give the PR "all the facts" upfront. For example, a policy may require the radio to sense in a certain band with a certain resolution, or to detect its location using GPS. The radio is de-coupled from the policies, so it does not "know" what they require. The radio also cannot constantly sense all bands or perform other costly operations (in terms of e.g. time or battery power). The ability to get constraints back from the PR means, in effect, that the reasoner tells the radio what it needs to do in order to satisfy current policies.

III. Approach

We approached the problem as a *constraint simplification* problem. As in most policy approaches, policies are constraints (in this case on radio behavior). The radio sends a request consisting of a set of *facts* to the PR. The PR tries to simplify (or reduce) the policies, using the provided facts, to the fact that the transmission is permitted. Sometimes the PR stops half-way there, because the provided facts are not sufficient to reduce all its constraints. In such cases, the remaining constraints are sent back to the radio, as explained above.

The other important aspect of our approach is that it is a mostly *functional* approach (as opposed to the *relational* approaches of e.g. OWL[2] and Prolog). There are several reasons for this,

- The requirement to reason about functions, such as powermasks.
- Constraint simplification works better with a functional approach than with a relational one, because with a functional approach we can use efficient *term rewriting* techniques as opposed to general reasoning.
- Having functions, defined by equations, allows us to express computations (such as temporal and geospatial ones) *inside* the language. In a relational approach, we would be forced to rely on procedural attachments, or some other ad-hoc solution, which would make policies very difficult to accredit and reason about.

Our language is not a domain-specific one. We have developed a general logic, based on a typed First-Order Logic, which is sufficient for the XG problem. However, the domain-specific parts are expressed in *ontologies*. By adding more ontologies, our language can be used for other types of policies. Ontologies will be discussed in Section IV-B.

CoRaL is set apart from other policy languages, such as KaOS, Rei, or Ponder [2], by being a (mostly) functional language based on constraint simplification, as required to express and reason about the kinds of policies we deal with in the spectrum domain. If one were to compare event-condition-action rule policy languages with CoRaL, one could interpret the radio's request as the event that triggers policy evaluation. Policies describe the conditions under which a transmission is allowed or not allowed. The PR answers with either granting or denying the use of spectrum to the requesting radio or telling the radio what it has to do to gain access. Possible action are therefore transmitting or performing further actions to satisfy constraints or making a new request.

IV. The CoRaL Language

Due to space limitations, we can only give a brief, informal presentation of the CoRaL language in this paper. For full details see [3].

A. Syntax

The building blocks of CoRaL are types, terms, and formulas.

Types can be either user-defined or one of the built-in types `Int`, `Float`, `Bool`. User-defined types can be either atomic types or constructed using the builtin type constructors for list (`[]`), tuple (`()`), set (`{}`), function (`->`), and predicate (`Pred`) types. Types are declared using the `type` keyword. One can also declare subtypes (using

[2] `http://www.w3.org/TR/owl-features/`

the `subtype` keyword) and equivalent types (using the `deftype` keyword).

Terms belong to types. Terms can be constants (which includes functions), or primitive values, such as integers or booleans. The primitive values, arithmetic operations (`=,-,*,/,mod`), and predicate symbols for equality, inequality, and orderings (`=, =/=, <, =<, >, >=`) are built-in. Other constants must either be declared using the `const` keyword, or constructed using one of the builtin term constructors for lists, sets and tuples, or using a user-defined constructor function. A term can be declared to be equivalent to another term using the `defconst` construct.

Constant and type declarations can have an optional `public` attribute, making them visible and useable in other policies.

Formulas are built from atomic formulas, i.e. a predicate constant applied to zero or more terms. These can be combined using propositional connectives (`and, or, implies, not, if`) and quantifiers (`forall, exists`).

There are three kinds of statements in CoRaL: Declarations (covered above), rules, and `use` statements. Rules are universally quantified formulas of certain forms. There are Horn-clause-like rules, e.g.

```
(forall x,y:Int) p(x,y) if q(x,y) or r(y,x);
```

and equational rules, e.g.

```
(forall x,y:Int) f(x) = g(y) if x+y = 10;
```

In both cases, the conditional `if` part is optional. The `use` keyword is used when a policy needs to use another policy or ontology.

A `policy` contains any number of statements, see Figure 2. Specifically, a policy has rules for two special predicates, `allow` and `disallow`. If `allow` can be proved, this policy allows the transmission, and if `disallow` can be proved, the transmission is prohibited. Note that we do *not* have the axiom `allow iff not disallow`. In fact, it is quite common for the same transmission to be both allowed by some policy and disallowed by another. To find out whether the transmission is permitted or not, we have a meta-rule "permit iff allowed by some policy and not disallowed by any policy". This means that disallowing policies override allowing ones. We could easily change the meta-rule to allow prioritized policies, but there was no need for this.

An `ontology` is a special case of a policy with no rules.

Policy Start	policy POLICYNAME is
Policy Imports	include …
Type Declarations	type … deftype …
Constant Declarations	const … defconst …
Rule Declarations	allow if … disallow if …
Policy End	end

Fig. 2. Components of a Policy

B. Ontologies

Rather than creating ontologies of the traditional kind, we use the more flexible approach of algebraic specifications and algebraic data types (ADTs) [4]. This allows us to not only state static facts about the domain, but to also express more dynamic, or computational, aspects. Examples are given below. However, in order to support standard ontology development and tools for OWL, we are currently extending our language with an OWL interface. This is possible since CoRaL is more expressive than OWL.

For example, for periods of time we introduce the ADT TimePeriod,

```
public type TimePeriod;
public const tp :
      TimeInstant, TimeInstant -> TimePeriod [ctor];
```

A TimePeriod is characterized by two TimeInstants (its start and end times). The function `tp` is a constructor, which means that any time period with the same start and end times are in fact the same time period. In order to be able to retrieve the start and end times of a time period, we need some operations,

```
public const startTime : TimePeriod -> TimeInstant;
public const endTime : TimePeriod -> TimeInstant;

(forall ?ps,?pe:TimeInstant)
startTime(tp(?ps,?pe)) = ?ps;

(forall ?ps,?pe:TimeInstant)
endTime(tp(?ps,?pe)) = ?pe;
```

First, the operations are declared (as functions on TimePeriods), and then they are defined using equations. We can also define more comples operations. For example, we need the ability to check whether a TimeInstant is within a TimePeriod,

```
public const inTimePeriod :
             TimeInstant, TimePeriod -> Bool;

(forall ?ti : TimeInstant, ?tp : TimePeriod)
inTimePeriod(?ti,?tp) = true if
   timeAfterOrSame(?ti,startTime(?tp)) = true and
   timeBeforeOrSame(?ti,endTime(?tp)) = true;
```

The `inTimePeriod` function is defined by an equation which depends on the previously defined `startTime` and `endTime` operations, as well as a couple of other operations defined elsewhere.

Since ADTs can be subtypes of other ADTs, we can define a type hierarchy, much like a traditional class hierarchy of ontology languages.

Using this approach, we have developed a basic set of ontologies for the spectrum domain, see Figure 3.

C. Policy Examples

The CoRaL language can best be understood by looking at some examples.

Allow to transmit in the band 5180 MHz to 5250 MHz, if the radio is at most 10km away from the geographic coordinates 39 10' 30" N, 75 01' 42", and only between 06:00 and 13:00 local time,

```
policy p1 is
  use request_params;
  allow if
    centerFrequency(req_transmission)
                  in {5180.0 .. 5250.0} and
    (exists ?le:LocationEvidence)
      req_evidence(?le) and
      distance(location(?le),loc1) =< 10000 and
    (exists ?te:TimeEvidence)
      req_evidence(?te) and
      hour(timeStamp(?te)) in {6 .. 12};
end
```

The `use` statement imports the `request_params` ontology, which defines the request parameters used here (see Figure 3). `req_transmission` is the requested transmission, which in this case must be within the given range. `{ .. }` is the term constructor for sets (ranges). For sensed data about the environment, such as time, location, and spectrum status, we have the notion of evidence, which must be presented to the reasoner.

We can combine the policy above with a restrictive policy, *Prohibit transmission if peak sensed received power is more than -80 dBm*

```
policy p2 is
  use request_params;
  disallow if
  (exists ?se:SignalEvidence)
    req_evidence(?se) and
    peakRxPower(?se) > -80.0;
end
```

It should be clear that the conditions for both policies can be true at the same time. In such cases, the meta-rule would make the second policy take precedence, and the transmission would not be allowed.

D. Semantics

CoRaL has a model-theoretic semantics very similar to that of classical first-order logic [5], i.e. based on set theory. Types denote sets and terms denote elements, in the usual way. There are some special cases, like the builtin lists, sets, and tuples, which denote finite sequences, sets, and tuples, respectively. As usual, there is also a notion of validity of formulas.

CoRaL also has an operational semantics, i.e. a set of proof rules. This includes rules for statically type-checking policies. The operational semantics is still under development, but the goal is to prove it sound with regard to the model-theoretic semantics.

V. Reasoning in CoRaL

Our first prototype policy reasoner, described in detail in [6] was implemented in Prolog. This implementation had a major limitation: It only gives yes/no answers to transmission requests, rather than returning unsatisfied constraints, as we have described above.

There are some basic technological requirements for a reasoner that can return constraints. While similar to a classical theorem-proving problem, there are several features that distinguish the XG problem.

Interactivity. Permission to transmit, when not immediately provable, can be obtained by a series of modified requests that result from interactions between the radio and the policy reasoner. For example, there may be a requirement to perform a sensing action which the radio has not yet performed. The radio can perform the action and submit a new request with additional sensing evidence. The evidence provided by the radio would increase monotonically until the interactive proof attempt succeeds or fails.

Anytime solutions. If the policy reasoner times out, the intermediate result, representing the proof state of the reasoner when it was interrupted, should be interpretable so that appropriate additional facts can be provided by the radio.

Predictability. Policy authors must be able to predict the behavior of the policy reasoner when given one of their policies, which is not usually a requirement for automated theorem provers. Thus, a clear operational semantics is needed and ad-hoc automated reasoning techniques that are sensitive to minor modifications should be avoided. Predictability refers not only to the final result but also to the time required. Thus, equational rewriting and certain logic-programming techniques are preferable to exhaustive search, because their dynamics can be better controlled.

Underspecified requests. A cognitive radio sometimes cannot or does not wish to form a fully specified transmission request. The radio may not be aware of

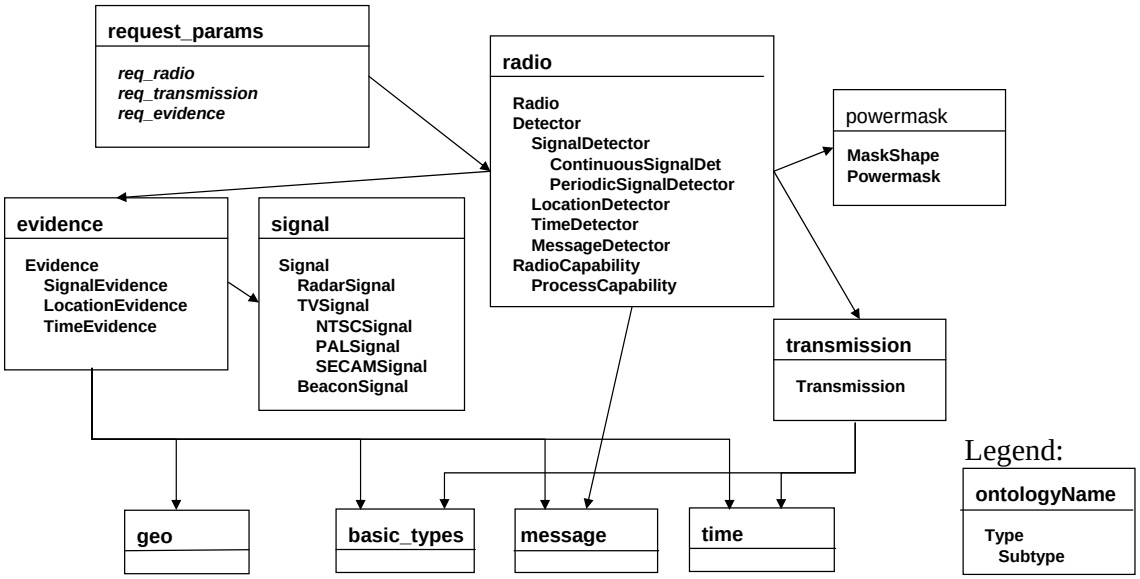

Fig. 3. Example Ontologies. Each box represents an ontology. The name on the top of the box is the name of the ontology. An arrow between two ontologies indicates that the ontology at the tail of the arrow uses the ontology at the head. For brevity, we only list types (in arial font) and constants (in italic font) in some of the ontologies. Functions, predicates, and axioms are not shown. Type hierarchies are represented using indentation.

all the applicable policies or may employ a strategy of not initiating costly sensing operations unless required. The policy reasoner tells the radio that the underspecified request would be valid if certain constraints were met. Thus, the policy reasoner should combine efficient evaluation of fully instantiated requests with reasoning about the more complicated constraints of underspecified requests.

Traditional, e.g., resolution-based theorem proving is too expensive as a solution and will not result in a fully automated policy reasoner of reasonable performance. More efficient techniques, such as pure equational or logic programming are not expressive enough to capture the reasoning steps required for CoRaL, since our language is rather expressive. Therefore, we explored the middle ground between these two extremes.

Using Maude [7], a language based on rewriting logic [8], we have specified the *proof system* of the policy reasoner. This prototype implementation uses a novel combination of functional, equational, and (constraint) logic-programming languages and of automated theorem proving [9], [10]. The proof system has four kinds of proof rules: Forward chaining, backward chaining, partial evaluation based on conditional rewriting, and constraint propagation and simplification.

The result is a *formal executable specification* that allows us to experiment with, and operationally understand, the unique combination of proof rules in the context of small policy examples. Efficiency was not a concern for the specification, which instead expresses proof rules in a way that is close to a mathematical presentation and also facilitates the proof of logical soundness. The specification is sufficiently high-level to serve as a reference for future implementations.

ACKNOWLEDGMENT

This research was supported by DARPA's neXt Generation (XG) Communications Program under Contract Numbers FA8750-05-C-0230 and FA8750-05-C-0150.

REFERENCES

[1] "ETSI Standard EN 301 893 V1.2.2 (2003-06)," 2003, reference DEN/BRAN-002000-2. [Online]. Available: http://www.etsi.org

[2] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok, "Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder," in *2nd International Semantic Web Conference (ISWC2003)*. Springer-Verlag, 2003.

[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., 2007.

[4] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 1*. Springer-Verlag, 1985.

[5] M.-O. Stehr, "Towards a universal policy logic," 2007.

[6] G. Denker, D. Elenius, R. Senanayake, M.-O. Stehr, and D. Wilkins, "A policy engine for spectrum sharing," SRI International, Tech. Rep., 2007, www.csl.sri.com/users/denker/Paper/2007.

[7] M. Clavel, F. Durán, S. Eker, J. Meseguer, and M.-O. Stehr, "Maude as a formal meta-tool," in *FM'99 — Formal Methods*, ser. Lecture Notes in Computer Science, J. Wing and J. Woodcock, Eds., vol. 1709. Springer-Verlag, 1999, pp. 1684–1703.

[8] J. Meseguer, "Conditional Rewriting Logic as a unified model of concurrency," *Theoretical Computer Science*, vol. 96, no. 1, pp. 73–155, 1992.

[9] A. Bundy, "A survey of automated deduction," *Lecture Notes in Computer Science*, vol. 1600, pp. 153–174, 1999. [Online]. Available: citeseer.ist.psu.edu/bundy99survey.html

[10] J. A. Robinson and A. Voronkov, Eds., *Handbook of Automated Reasoning*. Elsevier and MIT Press, 2001.