Ontology-based Spectrum Access Policies for Policy-based Cognitive Radios

Behnam Bahrak∗†, Jung-Min “Jerry” Park∗, Hao Wu†
∗Department of Electrical and Computer Engineering, Virginia Tech
{bahrak, jungmin}@vt.edu
†Department of Computer Science, Virginia Tech
wuha0723@vt.edu

Abstract—To maximize their efficacy, cognitive radios (CRs) need to be able to cope with the constantly changing spectrum environment, evolving spectrum access policies, and a diverse array of network application requirements. Policy-based cognitive radios address these challenges by decoupling the spectrum access policies from device-specific implementations and optimizations. These radios can invoke situation-appropriate, adaptive actions based on policy specifications and the current spectrum environment. A policy-based CR has a reasoning engine called a policy reasoner. The primary task carried out by the policy reasoner is evaluating the transmission requests with respect to the spectrum policies. In this paper, we describe the design of a policy reasoner that processes ontology-based spectrum policies. The main advantage of using ontology-based policies is that the policy reasoner can understand and process any spectrum policies authored by any organization by relying on the spectrum ontologies. In our implementation, the spectrum ontology defines the various dynamic spectrum access (DSA) concepts, models the domain of DSA networks in a machine-understandable manner, and uses SWRL (Semantic Web Rule Language) rules to represent spectrum policies. Unfortunately, ontological reasoning needed to process ontology-based spectrum policies incurs greater computation overhead compared to non-ontology-based reasoning. This drawback can be a critical one as it can impede a CR from meeting its real-time performance requirements. We have carried out a number of experiments, using our implementation, to evaluate whether a radio controlled by ontology-based policies can meet its real-time performance requirements. Based on our experimental results, we propose a set of guidelines for the design of ontology-based spectrum access policies.

I. INTRODUCTION

Fueled by the emergence of new wireless applications, the constantly increasing demand for more spectrum has resulted in the overcrowding of the spectrum that is appropriate for communications. New spectrum access technologies, strategies, and regulatory paradigms are needed to address this problem. Regulators, researchers, manufacturers, and other stakeholders are currently studying new spectrum access strategies, such as dynamic spectrum access and opportunistic spectrum sharing, to alleviate the spectrum shortage problem. The cognitive radio is seen as one of the key technologies for enabling new spectrum access strategies and utilizing the spectrum more efficiently. A cognitive radio, often implemented as a software-defined radio (SDR), autonomously configures its radio-system parameters to deliver the required quality-of-service (QoS) subject to an appropriate combination of device and operational limitations, user requirements, and regulatory policy constraints. In order to fully support more sophisticated and efficient spectrum access strategies, cognitive radios need to be able to cope with the constantly changing spectrum environment, evolving spectrum access policies, and a diverse array of network applications performance requirements. Policy-based cognitive radios address these challenges by decoupling the policies from device-specific implementations and optimizations. This type of radios can invoke situation-appropriate adaptive actions based on policy specifications and the current spectrum environment.

In order to fully support the new spectrum access strategies, cognitive radios need to be able to cope with evolving spectrum access policies and constantly changing application requirements. Policy-based cognitive radios address these challenges by decoupling the policies from device-specific implementations and optimizations. This type of radios can invoke situation-appropriate adaptive actions based on policy specifications and the current spectrum environment.

In order to regulate and enforce proper transmission behavior, policy-based cognitive radios need mechanisms to enforce spectrum access policies prescribed by the regulators. Most of these mechanisms are carried out by specialized software modules called policy conformance components (PCCs), which include the policy reasoner, policy enforcer, policy manager, and the policy database [18]. The policy reasoner (a.k.a. policy engine) carries out vital policy inference computations and is one of the most critical components of the PCCs. The primary task of the policy reasoner is to evaluate the transmission strategies of the radio against regulator-authorized policies to verify whether the strategies conform to the policies. This reasoning process is the first critical step in enforcing policy conformance of the cognitive radio’s transmission behavior.

In this paper, we describe the design and implementation of a policy reasoner that processes ontology-based spectrum policies. The main advantage of using ontology-based spectrum policies is that the policy reasoner can understand and process any spectrum policies authored by any organization by taking advantage of the spectrum ontologies. Using ontology-based spectrum policies has several other advantages, such as facilitating policy management, flexible knowledge...
representation, interoperability, flexible querying, and self-awareness, which we discuss in Section III. Unlike simple rule-based policy reasoners such as BRESAP that predefine the attributes of the spectrum policies and hard-wires them into the spectrum policy reasoner, our ontology-based policy reasoner can process policies whose attributes have not been predetermined in advance, without modifying the reasoning software itself.

In our implementation, the spectrum ontology defines the various DSA (dynamic spectrum access) concepts, models the domain of DSA networks in a machine-understandable manner, and uses SWRL (Semantic Web Rule Language) rules to represent spectrum policies. Our policy reasoner implementation is able to handle all ontology operations, including ontology-consistency checking and ontology information editing (i.e., adding, deleting, and updating). The policy reasoner uses a tableau-based algorithm [1] to carry out ontology reasoning, and the Rete algorithm to process spectrum policies and evaluate transmission requests. We modified the Rete algorithm so that it can compute spectrum opportunity constraints. Our implementation employs GNU Radio [5] and Universal Software Radio Peripheral (USRP) [11] to generate, send, and receive the signal waveforms.

The contributions of our work can be summarized as follows:

- We describe the design and implementation of a policy reasoner that processes ontology-based spectrum access policies. The policy reasoner uses the spectrum ontologies, the transmission strategy, and the spectrum policies to determine the legality of the transmission strategy and compute the corresponding opportunity constraints, if needed.
- One of the greatest challenges in designing a policy reasoner is devising an algorithm for efficiently determining whether a given transmission request conforms to a set of active ontology-based spectrum policies and, if needed, compute the spectrum opportunity constraints. We have met this challenge by devising an efficient algorithm that utilizes a Rete network to carry out the aforementioned task.
- Using our implementation, we have carried out a number of experiments to evaluate the efficacy of ontology-based policy reasoning for cognitive radios. Based on the evaluation results, we provide insights and guidelines for designing ontology-based spectrum access policies.

The rest of this paper is organized as follows. We first provide related technical background knowledge in Section II. Section III discusses advantages and challenges of using ontologies in the context of cognitive radios. In Section IV, we describe the detailed system design. System prototype implementation issues are discussed in Section V. In Section VI, we describe the experiments that were performed using our cognitive radio implementation and provide the results of the performance evaluation. Related work and conclusion are given in sections VII and VIII, respectively.

II. TECHNICAL BACKGROUND

A. OVERVIEW OF POLICY-BASED COGNITIVE RADIOS

In this subsection, we briefly introduce the architecture of a policy-based software defined cognitive radio system. The components of the policy-based cognitive radio system can be classified into server side components and client side components. At the server side, the main components of the cognitive radio system are the policy manager and the policy database [17]. The policy database stores all the spectrum policies authorized by the regulators (e.g., FCC). The policy manager is responsible for carrying out various policy data operations, including adding, deleting, and updating policies for a particular region; activating and deactivating spectrum policies; and pushing active spectrum policies to the cognitive radio client.

At the client side, the policy-based cognitive radio system consist of four components: sensors, radio front end, system strategy reasoner and policy reasoner. The functionalities of these four components are described below.

- Sensors: Enable the radio to sense its surrounding environment and determine the available spectrum opportunities;
- Radio Frontend: Radio uses radio frontend to transmit and receive RF signals;
- System Strategy Reasoner (SSR): The SSR controls the hardware, gathers sensory information, and formulates transmission strategies. The SSR interacts with the policy reasoner to determine the available spectrum opportunities that conform to the policies. Specifically, the SSR formulates a transmission strategy, based on collected sensory information and its current state, and sends this information to the policy reasoner in the form of a transmission request.
- Policy Reasoner (PR): The PR receives the transmission requests generated by the SSR and evaluates them for policy conformance. The result of the evaluation is returned to the SSR as a transmission reply along with a set of opportunity constraints, if applicable.

The XG (neXt Generation) communication program is a technology development project sponsored by DARPA’s Strategic Technology Office, with the goal of developing both the enabling technologies and system concepts to dynamically redistribute allocated spectrum along with novel waveforms [6]. The XG radio is a good example of a policy-based cognitive radio. Figure 1 illustrates the architecture of the XG radio.

The SSR and the PR are the reasoning engines of a cognitive radio. The SSR interacts with the policy reasoner to determine the available spectrum access opportunities that conform to the currently active set of policies. Specifically, the SSR formulates a transmission strategy, based on collected sensory information and its current state, and sends this information to the policy reasoner in the form of a transmission request. The policy reasoner evaluates the transmission request against the policies to verify whether the transmission strategy conforms
to the policies. If some of the configuration states in the transmission request are under specified (or invalid), then the policy reasoner needs to compute the missing parameters (or corrected parameters) and return them to the SSR along with a transmission reply that indicates transmission denial. The missing (or corrected) parameters computed by the policy reasoner are referred to as opportunity constraints.

The core reasoning problem in the policy-based cognitive radio is to infer from a given set of policies and a set of transmission strategies (contained in a transmission request), whether the transmission request should be allowed or denied; in addition, if denied, compute the opportunity constraints. The cognitive radio research community’s efforts to solve this challenging problem are ongoing.

B. Overview of Ontologies

Ontology refers to a system of categories that describe a particular vision of the world. Ontology relies on a certain philosophical view and is independent of natural languages. In computer and information sciences, ontology is a formal representation of knowledge and shared vocabulary about a particular domain of interest. It contains a set of concepts in that domain and relationships between these concepts, which can be used to describe and model this particular domain in a machine understandable manner [10]. Ontology is widely used in the realm of artificial intelligence, semantic web, system engineering, biomedical informatics, library science, etc.

A typical ontology consists of the following components:

- **Individuals**: Individuals are the basic components of an ontology. An individual in an ontology can be any concrete object or abstract object.
- **Classes**: A class is an abstraction of objects that have similar features. For example, Waveform is a class with individuals that are particular waveforms. Classes are organized into a hierarchy of classes that indicates relationships between the classes.
- **Properties**: Properties are attributes, features or parameters that an object or a class in an ontology can have, such as carrier frequency of a waveform, number of symbols in an alphabet, etc.
- **Relations**: Relations describe how objects can be related to other objects in an ontology. For example, a waveform represents a sequence of symbols from an alphabet; in this case, represent is a relation that links waveforms to symbol sequences.
- **Rules**: Rules are statements that describe the logical inferences that can be drawn from an assertion in the pattern of antecedent implicating consequent. For example, the following rule is a rule in the form of an if-then statement: “If a frame belongs to the SDLC (Synchronous Data Link Control) protocol, then the address field has 8 bits”.

When an actual ontology is built using the above components, it is important to check the consistency of the ontology and make sure there are no conflicts in the ontology. Tableau based reasoning algorithms are widely used to check the consistency of the ontologies. In such algorithms, a set of tableau expansion rules are applied to ontology data iteratively until no rule is applicable. Such operations can be time consuming, especially when the ontology is large. The details on the ontology reasoning process will be discussed later in Section IV.

III. ADVANTAGES AND CHALLENGES OF ONTOLOGY-BASED SPECTRUM POLICIES

Ontology-based policies rely largely on the expressive features of Description Logic languages, such as OWL (Web Ontology Language), to classify contexts and policies, thus enabling deductive inferences and static policy conflict resolution. In contrast, rule-based policies take the perspective of logic programming to encode the axioms and rules in a clear way [22]. Moreover, a rule-based approach facilitates the straightforward mapping of policies to lower-level enforcement mechanisms thanks to its concise and understandable syntax. But a simple rule-based policy reasoner (e.g., BRESAP [3]) has a number of shortcomings compared to a more complex ontology-based reasoner. For example, consider the following two policies:

**Policy 1.** Permit transmission in [255 MHz, 328 MHz], if the modulation type is GMSK.

**Policy 2.** Prohibit transmission in [255 MHz, 328 MHz], if the modulation type is FSK.

A rule-based policy reasoner would not detect a conflict between these two policies. But an ontology-based reasoner can recognize that GMSK (Gaussian Minimum Shift Keying) is a subclass of FSK (Frequency Shift Keying), so there is a conflict between these two policies. In other words, since an ontology-based reasoner is able to recognize subclass relations, it can identify and resolve policy conflicts that a rule-based policy reasoner cannot.

Using ontology-based spectrum access policies has several other advantages as discussed below.

**Policy management**: Ontologies can provide important supplemental information to simplify policy management and composition. In other words, ontologies facilitate the specification of complex regulatory policies for spectrum access.
Flexible knowledge representation: Ontologies can be used as the knowledge representation language for decision-making systems such as policy reasoners. This enables reuse of domain knowledge in the context of policies, and facilitates the analysis of domain knowledge. It is important for cognitive radios to represent knowledge so it can be used in the most flexible way, because they are required to react to the circumstances they have not seen before [14].

Interoperability: Ontologies facilitate the sharing of the policy structure among diverse software agents [15]. If different software agents use the same set of well defined terms for describing the domain and data of the spectrum access policies, it will be easier for them to “talk” to one another.

Flexible querying: Similar to humans, intelligent agents need to be able to answer queries from their users and other agents. For agents to formulate such queries, they must understand a common language, formally defined in syntax and semantics. Using ontologies, information can be queried, and such queries can be answered without having any explicit preprogrammed monitoring capability [14].

Self-awareness: A cognitive radio should be aware of its own capabilities and reflect on its own behavior. Using ontologies, radios can understand their own structure and modify their operation characteristics at runtime based on this understanding.

Flexible knowledge representation: Ontologies can be used as the knowledge representation language for decision-making systems such as policy reasoners. This enables reuse of domain knowledge in the context of policies, and facilitates the analysis of domain knowledge. It is important for cognitive radios to represent knowledge so it can be used in the most flexible way, because they are required to react to the circumstances they have not seen before [14].

Interoperability: Ontologies facilitate the sharing of the policy structure among diverse software agents [15]. If different software agents use the same set of well defined terms for describing the domain and data of the spectrum access policies, it will be easier for them to “talk” to one another.

Flexible querying: Similar to humans, intelligent agents need to be able to answer queries from their users and other agents. For agents to formulate such queries, they must understand a common language, formally defined in syntax and semantics. Using ontologies, information can be queried, and such queries can be answered without having any explicit preprogrammed monitoring capability [14].

Self-awareness: A cognitive radio should be aware of its own capabilities and reflect on its own behavior. Using ontologies, radios can understand their own structure and modify their operation characteristics at runtime based on this understanding.

As discussed above, ontology-based policy reasoning has a number of advantages when used for policy-based cognitive radios. However, it also introduces a number of challenges not shared by most of the other application areas of ontologies. The main challenge in using ontology-based spectrum access policies is meeting the real-time processing requirements of a cognitive radio. Real-time processing demands higher performance for inference and reasoning than an interactive application. In addition, the knowledge base of a cognitive radio includes state information that is continually varying. This is in contrast with the static knowledge bases employed by most ontology-based reasoning systems.

IV. SYSTEM DESIGN

In this section, we first briefly describe the design requirements of a policy-based cognitive radio that employs an ontology-based policy reasoner. Then, we introduce the architecture of our radio system and its core component, the ontology-based policy reasoner.

A. System Design Requirement

Cognitive radios have stringent real-time requirements. For example, XG radios need to satisfy the 100 ms channel evacuation time [18]. One way of satisfying real-time requirements is to tightly control the timing of software execution. Introducing ontology into the policy reasoning process will certainly cause additional runtime overhead and hinder the timing control of the software execution. This means that the ontology-based policy reasoner needs to be highly optimized to minimize the runtime overhead.

The design methodology of existing software defined radio (SDR) architectures suggest that software portability will likely be one of the primary requirements that will drive the hardware and software architectures of most software defined radios and cognitive radios. The importance of portability suggests that ontology-based policy reasoning systems that require support from specific hardware, operating system, or any other part of the computing system, other than the reasoning engine, may have limited utility.

Another design requirement of an ontology-based policy reasoner is the ability to handle a dynamic knowledge base that is continually varying. Unlike the other ontology-based reasoning systems that use static knowledge bases, an ontology-based policy reasoner needs to handle a continuously changing knowledge base.

B. Overall System Architecture

Figure 2 illustrates the architecture of the cognitive radio system that our ontology-based policy reasoner is a part of. Our policy reasoner was designed to work as a component of such a system. The system consists of six main components: radio hardware and software, sensors, system strategy reasoner, reasoning interface, ontology-based policy reasoner, and a remote ontology and policy server. The role of the system strategy reasoner in our system is the same as that in the XG radio, which was described in section II-A. The functionalities of the rest of the components are described below.

- **Radio Hardware and Software:** The radio software receives parameter configurations from the SSR, establishes the datapath and corresponding transmission parameters, and sends control message to the radio hardware in order to transmit the waveforms;
- **Reasoning Interface:** The reasoning interface receives transmission strategies from the SSR, translates them into a transmission request and sends it to the policy reasoner. The reasoning interface also receives the transmission replies from the policy reasoner and forwards them to the SSR;

![Architecture of the cognitive radio system.](image-url)
• **Ontology-based Policy Reasoner:** The ontology-based policy reasoner carries out all of the spectrum ontology and policy related tasks, including loading and processing spectrum ontologies and policies, checking the consistency of spectrum ontologies, etc. In addition, the policy reasoner evaluates the transmission request against the currently active spectrum policies to check its conformance with the policies. The results of the evaluation, in the form of a transmission reply (or transmission decision), is sent to the reasoning interface;

• **Remote Ontology and Policy Server:** The remote server stores and manages the spectrum ontologies and policies created by various spectrum regulatory entities.

When the cognitive radio is turned on, its components perform various initialization tasks. The ontology-based policy reasoner first loads the spectrum ontologies and the spectrum policies from the ontology and policy server. After loading is finished, the reasoner translates the spectrum ontologies and policies into forms amenable to processing, and then checks the consistency of the spectrum ontologies. After all of these initialization tasks have been executed, the radio is ready for transmitting waveforms.

When a waveform needs to be transmitted, the upper layer application sends the transmission requirements to the SSR. In response, the SSR collects spectrum availability information from the sensors and formulates a transmission strategy using the spectrum availability information and the transmission requirements. The SSR sends the transmission strategy to the ontology-based policy reasoner in the form of a transmission request via the reasoning interface. The policy reasoner checks whether the transmission request conforms to the loaded spectrum policies and then formulates a transmission reply (decision). If the transmission request conforms to the policies, then the policy reasoner generates a transmission reply indicating transmission approval; otherwise, it generates a reply indicating transmission denial. In the latter case, the policy reasoner also generates *spectrum opportunity constraints*—these constraints are specifications of missing transmission parameters (in case of an under-specified transmission request) or revised transmission parameters (in case of an invalid transmission request). If the transmission is approved, the SSR sends the transmission parameter configuration to the radio software, and the radio software sends control messages to the radio hardware to begin waveform transmission.

**C. Ontology-based Policy Reasoner**

We employ Pellet [21] as the ontology/policy reasoning engine with some modifications to enable the computation of opportunity constraints. Figure 3 illustrates the reasoner’s architecture. The policy reasoner has three main components: ontology and policy loader, ontology reasoner, and policy reasoner. In the rest of this section, we will describe each of these components and the algorithm for computing the spectrum opportunity constraints in detail.

1) **Ontology and Policy Loader:** The ontology and policy loader is responsible for loading spectrum ontologies and active spectrum policies from the remote spectrum ontology and policy server. After the ontologies and policies are loaded onto the radio, the ontology and policy loader parses them into internal data structures and separates the ontologies and the policies. The spectrum ontologies are fed into the ontology reasoner and the reasoner performs the necessary processing. Then the spectrum policies are sent to the policy reasoner and the policy-related processing is performed.

2) **Ontology Reasoner:** Once the ontology reasoner receives spectrum ontology information from the ontology and policy loader, it translates the ontology information into ontology facts and axioms. These ontology facts and axioms construct the knowledge base of the domain modeled by the spectrum ontologies. This knowledge base is fed to the tableau-based reasoner to check the consistency of the ontologies stored in the knowledge base. An ontology is consistent if there is an interpretation that satisfies every fact and axiom in this ontology. Such an interpretation is called a *model* of the ontology [21]. The tableau based reasoner runs the tableau-based reasoning algorithm that checks the consistency of ontologies. The tableau-based reasoning algorithm first constructs an initial completion graph from the knowledge base. The nodes in the completion graph represent individuals and literals in the ontology, and the directed edges between nodes represent the property-value assertions related to the individuals and literals in the ontology. Each node in the completion graph is associated with a certain type of an ontology class or a numeric data type. A set of tableau expansion rules is applied to the completion graph repeatedly until either a clash occurs or there exists no more tableau expansion rules to apply. The former case implies that the ontology is inconsistent, whereas the latter case implies that the ontology is consistent and a model of the ontology exists [21]. The tableau based reasoner also contains a XSD (XML Schema
Definition) datatype reasoner, which can reason with XSD datatypes [4]. During the tableau-based reasoning process, the XSD datatype reasoner is invoked to determine the consistency of the datatype when a literal node is found by the tableau-based reasoner in the completion graph. The details of the tableau-based reasoning algorithm for ontology consistency checking are beyond the scope of this paper, and we will not present them here.

Checking the consistency of the ontology is important, because if the spectrum ontologies are inconsistent, then conflicts will exist in the ontology definitions. When the policy reasoner evaluates the transmission requests based on such inconsistent spectrum ontologies, it may produce incorrect transmission replies. The ontology consistency checking needs to be carried out whenever there is a change to the spectrum ontologies. Checking the consistency of the ontology is important, because if the spectrum ontologies are inconsistent, then conflicts will exist in the ontology definitions. When the policy reasoner evaluates the transmission requests based on such inconsistent spectrum ontologies, it may produce incorrect transmission replies. The ontology consistency checking needs to be carried out every time the spectrum ontologies are changed.

3) Policy Reasoner: The policy reasoner evaluates transmission requests against the active spectrum policies using the Rete algorithm and related Rete networks. The Rete algorithm is an efficient pattern matching algorithm for implementing production rule systems. It sacrifices memory for increased execution speed. This algorithm constructs a data flow network (i.e., Rete network) to represent the rules [9].

A Rete network has three elements: working memory, alpha network and beta network:

- **Working Memory (WM):** The working memory stores all the asserted facts that will be evaluated against the rules represented by the Rete network. The working memory contains several working memory elements (WME), and each working memory element stores a single fact. In most of the cases, the asserted facts are represented by a triple, in the form of $<$ subject predicate object $>$. 

- **Alpha Network:** The alpha network of a Rete network is a discrimination sub-network that selects facts stored in WMEs. It carries out simple conditional tests on the fact attribute values against constant values or pattern matching tests on the patterns of facts stored in WMEs. Each node in an alpha network represents a certain condition or pattern in the rules. For example, if the pattern represented by an alpha node is $<$ ? color red $>$, then all the objects that are red pass the pattern test carried out by this alpha node, and they will be passed to subsequent alpha nodes if there is any. Different sets of alpha nodes are connected together to form different paths that represent different conditions described in the rules. Facts that pass all the alpha nodes in a path are stored in the corresponding alpha memory, which connects the alpha network and the beta network.

- **Beta Network:** The beta network consists of beta nodes and corresponding beta memories, and mainly performs join operations between different WMEs. Every beta node has two inputs. One input is connected to the corresponding alpha memory of a certain alpha node path (a condition described in the rules), and the other input is connected to other beta memories. Hence, beta nodes also perform the role of connecting different rule conditions (alpha node paths) together to form different rules. Each beta node performs the join operation to merge the WMEs coming from the two inputs into a single list of WMEs and stores it directly into the corresponding beta memory. This list of WMEs stores the facts that partially match the corresponding rule. Such join operations are carried out repeatedly by the beta nodes, and the resulting WME list travels the beta network in a downward direction. When the WME list reaches the terminal node of a rule, this means that all of the necessary conditions have been combined together to form the required rule and that the list contains all of the facts that satisfy that rule.

Spectrum policies have the same role as the rules in a production rule system. When the policy reasoner receives the spectrum policies from the ontology and policy loader, it first parses the spectrum policy rules into several policy conditions, and then, uses these policy conditions to construct the Rete network that is used later for evaluating transmission requests.

The following example shows the Rete network construction process for a simple spectrum access policy.

**Policy 3.** Allow transmission if the transmission center frequency is 240 MHz and the bandwidth is 10 MHz.

This spectrum policy is translated into an internal policy rule structure that has the following policy conditions:

- TransmissionRequest(?TranReqVar)
- CenterFrequency(?TranReqVar, ?ValueVar)
- Bandwidth(?TranReqVar, ?ValueVar)
- AllowTransmissionRequest(?TranReqVar)

The first three conditions constitute the body part of the spectrum policy rule, and the last condition corresponds to the conclusion part. Using these four policy conditions, a Rete network, which represents a policy, can be constructed. Each condition in the body part of the spectrum policy rule is represented by an alpha node in the Rete network. Beta nodes are used to connect all the alpha nodes to form the Rete network. The policy condition that corresponds to the conclusion part of the spectrum policy is placed in the last beta node, which acts as the terminal node of the Rete network. Figure 4 shows the resulting Rete network that represents the spectrum policy used in this example.

**4) Evaluating Transmission Requests:** The transmission request evaluation process is carried out by the ontology-based policy reasoner in two phases. In the first phase, the patterns of a transmission request are matched against the patterns of the spectrum policies using the Rete algorithm. The ontology-based policy reasoner treats a transmission request

\[^{1}\text{A string of characters that starts with a question mark is a variable}\]
as an ontology individual associated with the ontology class, TransmissionRequest, in the spectrum ontology. A transmission request has a set of properties, such as center frequency, bandwidth, transmission power, etc. These properties associated with the transmission request are each stored as a three-tuple in the ontology-based policy reasoner. When the pattern matching process begins, these three-tuples are fed to every alpha node in the alpha network to perform a pattern matching test. The tuples that pass the pattern matching test are stored in the corresponding alpha memory. A beta node connected to an alpha memory and previous beta memory performs join operations (i.e., combining) on the tuples stored in the two kinds of memories, and forwards the results of the join operations to the next beta node. The first phase of the transmission request evaluation process ends when the results of the join operations reach the last beta node (i.e., terminal node) and are non-empty. The individuals contained in the results match all the patterns described in the spectrum policies.

The completion of the first phase indicates that the patterns of the transmission request conform to the policy’s patterns. The second phase is needed to test whether the values of the transmission request properties (e.g., center frequency, transmission power, etc.) conform to the conditions prescribed in the spectrum policies. The ontology-based policy reasoner invokes particular data type converters to convert the transmission request property values into related data types, and compares these values to the values given in the spectrum policies. Once all the comparisons for the value conditions described in the spectrum policies have been completed, the legality of the transmission request can be determined.

When a transmission request is declined, the ontology-based policy reasoner computes the spectrum opportunity constraints in addition to the transmission reply. Whenever a pattern matching test in the first phase or a value comparison test in the second phase fails, the unsatisfied rule pattern and the corresponding spectrum policy are recorded. After the evaluation process is finished, the recorded spectrum opportunity constraints are sent to the SSR. A pseudo code of the algorithm for evaluating a transmission request is given by Algorithm 1.

### Algorithm 1: Transmission Request Evaluation

**Input:** An individual of a transmission request, \( T \).

**Output:** A transmission reply, \( D \), and opportunity constraints, \( List_{OC} \), if needed.

1. \( rete = \text{construct}_\text{rete}_\text{network}() \)
2. \( Set_{tuples} \): a set of tuples that describe \( T \)
3. \( Set_{tuples} = \text{compile}_\text{facts}(T) \)
4. **for** tuple in \( Set_{tuples} \), **do**
   5. \( \text{result} = rete.\text{match}_\text{pattern}(\text{tuple}) \)
   6. **if** \( \text{result} \neq \text{matched} \) **then**
      7. \( \text{rete.get}_\text{opportunity}_\text{constraint}(List_{OC}) \)
      8. \( D = \text{“not allowed”} \)
      9. **return**
   10. **else**
      11. \( \text{Set}_{\text{matched}}.\text{add}(\text{tuple}) \)
   12. **end** **if**
5. **end** **for**
6. **for** tuple in \( Set_{\text{matched}} \), **do**
   7. **if** tuple.\text{has}_\text{numerical}_\text{property}_\text{value}() **then**
      8. \( \text{result} = \text{compare}_\text{with}_\text{policies}(\text{tuple}) \)
      9. **if** \( \text{result} \neq \text{conform} \) **then**
         10. \( \text{get}_\text{opportunity}_\text{constraint}(List_{OC}) \)
         11. \( D = \text{“not allowed”} \)
         12. **return**
      13. **end** **if**
   14. **end** **if**
   15. **end** **for**
16. \( D = \text{“allowed”} \)
17. **return**

### V. System Prototype Implementation

We have implemented a prototype of a software-defined radio whose transmission behavior is controlled by ontology-based spectrum policies. In this section, we discuss the implementation of the critical components of the radio system.

#### A. Spectrum Ontology and Policy Documents

The spectrum ontologies are implemented using OWL DL (OWL Web Ontology Language Description Logic) variant of standard OWL web ontology language [7] in the OWL/RDF syntax. The OWL web ontology language is a semantic markup language based on XML (eXtensible Markup Language) that can represent and share ontologies on the Web. OWL is an extension of RDF (Resource Description Framework) in vocabulary and it is advocated by W3C as a standard web ontology language.

The spectrum policies are written in the Semantic Web Rule Language (SWRL) [12]. SWRL combines OWL DL and OWL Lite variants of the OWL web ontology language with the Unary/Binary Datalog RuleML sublanguage of the Rule Markup language (RuleML). With the features of RuleML language, SWRL can express horn-like rules. Since SWRL also combines the OWL DL variant, the horn-like rules can...
The spectrum ontology and policy documents stored in the spectrum ontology and policy server are organized as follows. Spectrum ontologies defined by each spectrum regulatory agency are stored in a separate document, and the spectrum ontologies can refer to each other to construct complicated spectrum ontologies using the `import` feature provided by the OWL web ontology language. Spectrum policies created by different spectrum organizations are stored in separate policy documents, and the policy documents can also refer to spectrum ontologies to acquire ontology information using the `import` feature. The component-based structure described above facilitates the management of the spectrum ontologies and policies. Figure 5 illustrates how the spectrum ontology and policy documents interact with one another.

![Interaction between spectrum ontology and policy documents.](image)

**Fig. 5. Interaction between spectrum ontology and policy documents.**

### B. Ontology-based Policy Reasoner

We employed Pellet (version 2.1.1) as the core ontology reasoning engine. Pellet is an open source OWL DL reasoner, and is written in Java. Pellet provides sound-and-complete OWL DL reasoning and supports reasoning with ontology individuals, ontology queries, and OWL/Rule hybrid reasoning [7], [21]. One of the most important features of Pellet is that it adopts a number of optimization techniques for ontology reasoning to minimize the runtime overhead of ontology reasoning. Such a feature makes Pellet an attractive choice for use in a cognitive radio since a cognitive radio has strict real-time processing requirements. We modified parts of the Pellet’s source code to enable the policy reasoner to compute spectrum opportunity constraints.

### C. Waveform Generation and System Integration

We employed GNU Radio and USRP 1 boards to generate the waveforms and implement the radio frontend [5]. We used two USRP 1 boards, both connected to host computers via USB 2.0 ports, one as the transmitter and the other one as the receiver. Separate Python programs are used to control the transmission and reception of waveforms. Because our experiments focused on the evaluation of the ontology-based policy reasoner, we did not implement a full-blown SSR, and instead functionalities of the SSR were emulated by a Python program that interacts with the policy reasoner. This SSR implementation generates randomly chosen transmission strategies and sends them to the policy reasoner. In our implementation, the ontology-based policy reasoner is written in Java whereas the rest of the software defined portions of the radio system is written in Python and C++. Hence, we had to bridge the Java run-time and Python run-time environments. We used JPype to implement the Java-Python interface. JPype is a tool that enables full access to Java libraries from Python programs. Access to Java libraries in Python programs was implemented by interfacing at the native level in both Java and Python virtual machines [16]. JPype can a start Java virtual machine in the Python run-time environment so that Java libraries can be invoked. In our radio system, the ontology-based policy reasoner is packaged into a Jar file and invoked from the SSR program. The communication between the SSR and the policy reasoner flows through JPype. Data type conversion from Python data type to Java data type and vice versa is automatically handled by JPype.

### VI. Experimental Evaluation

Our radio system prototype executes in two phases: preparation phase and transmission phase. The preparation phase is executed before waveform transmission. These two phases are described below:

- **Preparation phase:** In the preparation phase, the radio carries out all of the initialization tasks of the radio system, which include loading the spectrum ontology information, executing initial ontology consistency checks, loading currently active spectrum policies, and preparing a Rete network using the spectrum policies. The tasks carried out in the preparation phase are not time critical because those tasks are completed before a transmission request is evaluated.

- **Transmission phase:** In the transmission phase, the radio evaluates a transmission request generated by the SSR and transmits the corresponding waveform if the transmission request is allowed by the policy reasoner. The runtime performance of this phase affects the performance of the upper layer applications. The tasks carried out in the transmission phase are time critical and need to satisfy the radio’s real-time latency requirements (e.g., channel evacuation time).

In our experiments, we used the spectrum ontology and related spectrum policies created by SRI International [24]. The syntax of the SRI’s ontology and policies were modified so that they can be understood and processed by our policy reasoner. SRI’s spectrum ontology contains 40 classes, 75 properties and 24 individuals.

All the experiments were carried out on a PC with an Intel Core i5 2.67 GHz CPU, 4 GB memory, and running Ubuntu 10.04 operating system. It is obvious that the processing time of our policy reasoner implementation is dependent on the computing platforms computing power. To
obtain normalized time measurements that are independent of the platforms computing power, we divided all of the measurements with a baseline value. The baseline for each experiment is different, and they are explained in the following paragraphs. In all of the time measurements, we calculated the average of 20 independent runs.

A. Preparation Phase Evaluation

In this section, we describe the results from two experiments that were performed to evaluate the performance of the preparation phase. In the first experiment, we evaluated the impact of the spectrum ontology’s size on the ontology loading and ontology consistency checking time. In the second experiment, we investigated the effect of the active spectrum policies’ size on the Rete network construction time. The detailed descriptions of these two experiments and their results are described below.

1) Impact of Ontology Size on Loading and Consistency Checking Time: In this experiment, we only consider the impact of the number of individuals and the complexity of the individuals on the ontology loading and consistency checking time. We randomly generate individuals for a number of ontology classes and vary the total number of individuals in the spectrum ontology from 20 to 120. We also change the complexity of the individuals, which is quantified by the number of properties of an individual, in the spectrum ontology. The baseline for this experiment is a case where the number of individuals is 20 and each individual has zero property. The result of this experiment is shown in Figure 6. In this figure, ontology setup time denotes the time required to perform ontology loading and initial ontology consistency checking tasks.

From Figure 6, we observe that increasing the complexity of individuals (i.e., increasing number of properties per individual) has a noticeable impact on the growth rate of the ontology setup time. When the individuals are simple (i.e., have a small number of properties), increasing the number of individuals causes the setup time to increase at a moderate rate; when the individuals are more complex (i.e., have a larger number of properties), the setup time increases at a faster rate.

2) Impact of Policy Size on Rete Network Construction Time: The Rete network is constructed from the active spectrum policies loaded into the radio system. Active policies are the specific policies that are considered when evaluating a transmission request. When measuring the Rete network construction time, we vary the number of loaded policies and their complexity. The metric for a policy’s complexity is the number of conditions in it. The results of the experiment are given in Figure 7.

The baseline for this experiment is a case in which 5 active policies are loaded and each policy has 3 conditions. From Figure 7, we observe that the Rete network construction time is proportional to the number of loaded policies and their complexity.

B. Transmission Phase Evaluation

The most computationally expensive task performed in the transmission phase is the evaluation of the transmission request. Therefore, this task has a significant effect on whether a radio can satisfy its critical timing requirements, such as the channel evacuation time. The channel evacuation time is the maximum amount of sojourn time in the current channel after the detection of a primary user signal and before hopping to a fallow channel. We performed a number of experiments to measure the runtime overhead of the transmission evaluation process.

1) Impact of Active Spectrum Policies: There are two policy attributes that affect the runtime overhead of a transmission request evaluation: number of active policies and their complexity. Note that the complexity of the policies affects the size of the Rete network. We can classify the conditions of spectrum policies into two categories:

- **Pattern Conditions**: Conditions that match the pattern of ontology individuals with the spectrum policies;
• **Numeric Conditions**: Conditions that compare the values of the ontology individual properties with the values given in the spectrum policies.

In the first set of experiments, we vary the number of active spectrum policies from zero to 15. The zero policy case represents a scenario in which a transmission request is allowed without any policy-conformance checking, and this case serves as the baseline. We also vary the complexity of the spectrum policies. For each instance, the time required to evaluate the same single transmission request is measured. Figure 8 shows the results of these experiments.

As expected, the time required for transmission request evaluation increases proportionally to the increase in the number of loaded spectrum policies. From the figure, we can observe some interesting results related to policy types. By comparing the four curves in Figure 8, we observe that the number of numerical conditions (in a policy) has a noticeable impact on the transmission request evaluation time, whereas the number of pattern conditions has a minor impact. This phenomenon can be explained by the fact that the policy reasoner needs to perform data type conversion operations and data value comparison operations when processing numeric conditions, and these are computationally costly operations.

2) **Impact of Spectrum Ontology Size**: Our ontology and policy reasoner uses a Rete pattern matching algorithm to evaluate a transmission request against spectrum policy rules. In the reasoning process, the reasoner examines all of the individuals in the spectrum ontologies against the spectrum policy rules while ignoring the classes and properties of the ontologies. Therefore, we expect that the number of individuals in the ontologies to have a direct impact on the transmission request evaluation time, whereas the number of classes and properties to have no or minor impact. We ran two sets of experiments to study the effect of ontology size on the transmission request evaluation time. In both sets of experiments, the baseline case is a case in which there are 20 individuals in the ontology and each individual has zero properties.

In the first set of experiments, individuals in the spectrum ontologies do not match any of the rule patterns in the policies. The results are shown in Figure 9. From the figure, we can observe that the complexity of the ontology (i.e., number of properties per individual) has a minor impact on the transmission request evaluation time. This phenomenon can be explained by the fact that the Rete algorithm does not evaluate the individuals of the ontology because they do not match any of the policies’ rule patterns. Hence, an increase/decrease in the number of properties per individual does not affect the runtime overhead of the transmission request evaluation.

In the second set of experiments, individuals in the spectrum ontologies do match some of the rule patterns in the policies. The results are shown in Figure 10. From the figure, we
observe that the complexity of the ontology (i.e., number of properties per individual) does have a direct impact on the transmission request evaluation time. This phenomenon can be explained by the fact that every individual in the spectrum ontology that matches the rule patterns of the spectrum policies is inserted into the Rete network, and the Rete algorithm checks whether this individual conforms to the spectrum policies. These operations performed by the policy reasoner cause additional runtime overhead, and this overhead increases proportionally with the increase in ontology complexity.

C. Guidelines for the Design of Ontology-based Policies

Through a number of experiments, we have investigated the impact of several factors—including the size and complexity of the spectrum ontology and the spectrum policies—on the runtime overhead of the preparation phase and the transmission phase. Here, we share a number of insights that we were able to gain through our experimental results. We provide these insights in the hopes that they may serve as useful guidelines on how to design spectrum ontologies and policies for cognitive radio networks.

1) According to Figure 10, when spectrum access policies are involved, there is an inherent tradeoff between the complexity of spectrum ontologies and the runtime overhead of the ontology/policy reasoner. More complex ontologies enable spectrum regulators to author more descriptive and detailed policies but increase the processing burden of the ontology/policy reasoner.

2) The experiment results that are illustrated in Figure 8 imply that in order to minimize the transmission request evaluation time, spectrum policies need to be authored in such a way that their number of numeric conditions is minimized. These results show that the number of pattern conditions does not have a noticeable effect on the transmission request evaluation time.

3) The spectrum ontology should be designed in such a way that the number of pattern matches between the ontology’s individuals and the spectrum policies is minimized. According to Figure 9, this reduces the transmission request evaluation time significantly.

VII. RELATED WORK

In the past few years, policy-based network management and access control have attracted significant attention, and new proposals for policy languages and policy-based network management schemes and architectures have been put forth.

In [20], Riihijarvi et al. present their work on extending the current policy languages to a wider range of context, not limited to the spectrum domain. The authors believe that in the policy-based cognitive radio, not only spectrum policies but other non-spectrum related policies require management, and the interplay between spectrum and non-spectrum policies will influence the final reasoning result. They analyze the requirements of extending the policy language to non-spectrum areas, and their system prototype implementation shows that a rich policy language will benefit the operators.

DARPA’s neXt Generation (XG) communication program proposes a framework for the policy-based cognitive radio system [6]. Based on the XG radio architecture, companies such as SRI International [8] [23] and Shared Spectrum Company [17] [18] have proposed policy based architectures for dynamic spectrum access control. In [17] and [18], besides policy reasoner and reasoning process, policy database and policy management are also discussed. The policy database can be located inside the local cognitive radio or placed at remote side as a server. In [8] and [23], Denker et al. propose a policy language called CoRaL, and introduce the concept of ontology into their policy engine. However, no ontology reasoning services (consistency checking, etc.) are provided in their system, and the Prolog-based reasoning engine also limits the completeness of the reasoning process. The MITRE Corporation [19] also designed an ontology-based reasoning architecture for context-aware radios. However, no transmission request evaluation process is discussed in their paper and the computational overhead of ontology-based reasoning is not considered. None of the works discussed above address the issue of computing the opportunity constraints.

In [3] and [2] we introduced BRESAP, a multi-terminal binary decision diagram (MTBDD) based policy reasoner for processing spectrum access policies. BRESAP translates each constraint in every spectrum policy into a boolean expression and uses the MTBDD to organize these boolean expressions. MTBDDs are combined to form a single binary decision graph (BDD), and the policy reasoner uses this graph during the transmission request evaluation process. The major drawback of BRESAP is that it predefines the attributes of the spectrum policies and hard-wires them into the spectrum policy reasoner. BRESAP cannot process policies whose attributes have not been predetermined in advance. BRESAP can process such policies only after modifying the reasoning software itself.

IEEE P1900 working group (a.k.a IEEE SCC41) is a standard committee that aims to develop standards dealing with new technologies developed for next generation radio, advanced spectrum management and dynamic spectrum access [13]. There are six sub-groups within the IEEE P1900 working group, IEEE P1900.1 to IEEE P1900.6. Each of them makes their efforts to the standardization work in the different sub-areas of next generation communication radios and networks. IEEE P1900.5 sub-group is working on policy languages and policy architectures for managing cognitive radio for dynamic spectrum access applications. The goal of this working group is to develop a set of policy languages and policy architectures that specify cognitive radio behaviors and functionalities for dynamic spectrum access applications.

VIII. CONCLUSION

In this paper, we discussed the design and implementation of a spectrum policy reasoner that processes ontology-based spectrum policies for cognitive radio networks. In ontology-based spectrum access policies, attributes of the spectrum policies are decoupled from the design of the policy controlled cognitive radio and the policy reasoner, which enables
the policy reasoner to understand and process any spectrum policies authored by any spectrum regulators. Unfortunately, processing the spectrum ontologies and the ontology-based policies incurs additional computation overhead, which makes it more difficult to design a cognitive radio that meets the required real-time latency requirements. We have implemented a software-defined radio system, and carried out a set of experiments to evaluate the performance of the radio when it is controlled by ontology-based spectrum policies. Based on the insights gained from the experiments, we have also provided a set of guidelines for designing ontology-based spectrum policies.

REFERENCES