A COMPARISON OF XML AND OWL BASED APPROACHES TO REPRESENTING COGNITIVE RADIO FUNCTIONS

Yanji Chen (chen.yanj@husky.neu.edu)[1], Jakub J. Moskal (jmoskal@vistology.com)[2], Mieczyslaw M. Kokar (m.kokar@neu.edu)[1], and Kaushik R. Chowdhury (krc@ece.neu.edu)[1]

[1]Northeastern University, Boston, MA, USA
[2]VIStology, Inc., Framingham, MA, USA

ABSTRACT

A wireless network of software-defined radios can be considered as a distributed computing system, since the radios that are on the network possess the capabilities of performing various sensing and computation tasks requested by the applications running on other nodes. In order to achieve such an objective, radios need to perform various functions, often attributed to cognitive radios. E.g., devices may need to inform the network of their capabilities, applications may issue requests for services, and the network then can match the radio capabilities against the requests. Similarly, devices may request permission to transmit, which then need to be matched against policies and available resources (e.g., availability of spectrum). In all such scenarios, matching would have to be performed in order to derive decisions. One of the research questions is what languages are good for describing the requests, radio capabilities and policies? Since the solution to this problem needs to be flexible enough to address the scenarios in which radios with previously unknown capabilities and new applications can join the network dynamically, the language must be interpretable by the applications, radio devices and the network. In this paper we consider two options for expressing and solving the matching problem - an XML based approach and a Web Ontology Language approach. While our ultimate goal is to use quantitative metrics, such as processing time, bandwidth usage, precision, recall and F-measures to evaluate the two approaches, here we show how we approach this problem and our qualitative assessments of the two approaches based on our initial results.

1. INTRODUCTION

Wireless RF devices in a network need to interoperate in order to be able to perform their own tasks while not interfering with other network nodes. As such, devices need to be able to interpret ("understand") requests from other nodes and make decisions on what they can do without causing harm to other nodes. The progression of the capabilities of the RF devices towards *cognitive radios*, as advocated in [1], is expected to lead to more flexibility of the radios to understand the communication environment and own capabilities and making more "intelligent" decisions. In particular, it is expected that the interpretation capabilities will exceed the types hard-coded in the communications protocols of the particular layers of the ISO stack. The processing cycle of the cognitive radio follows the OODA (Object, Orient, Decide, Act) functionality pattern. While any radio can be said to follow such a pattern to a degree, a cognitive radio pushes the envelope by its ability to interpret the requests (external and own) that go beyond what is expected (here "expected" means hard-coded) from a specific communications protocol. This envelope is defined by the variety of the types of requests. While in a traditional radio this variety is dictated by the protocol, in a cognitive radio the variety is limited only by the expressiveness of the language that a radio can interpret. This paper provides a contribution to the investigation of the types of the languages that can be used by cognitive radios.

The interpretation process requires that information (e.g., requests) and communications environment (e.g., spectrum availability or device capabilities) be represented in a language. The requests can be understood as *queries*. Cognitive engines then use derivation rules (e.g., policies) as well as background knowledge in order to answer the queries, it is to derive what actions to take. The derivation process will *match* the preconditions of the derivation rules against the available information in the process of *inferring* decisions. Clearly, all the information that the matching process can operate on must be expressed in the language that the cognitive engine is capable of interpreting. The question of which languages are better for this kind of task has been an active area of research (cf. [2]).

One possible choice for such a language is to use the eXtensible Markup Language (XML) based technology, i.e., use XML schema definition (XSD) [3] as a base for expressing types, and then use XML to describe instance XML data about the environment and the resources. Application requests would then be expressed in XQuery [4] - the "native" query language for XML. The inference engine would then answer the queries by matching them with the XML instance data using an XQuery processor.

Another approach could use the Semantic Web based approach - the Web Ontology Language (OWL) [5] and the SPARQL query language. In this case an ontology [6] for

this domain would serve as the base for describing the instance data, the application requests/queries and answers to the queries. Application requests would be expressed in SPARQL and the matching would be performed by a SPARQL processor. Due to the fact that OWL has formal semantics, the data annotated in OWL can be processed by any *inference engine* (or *reasoner*) conformant with the OWL semantics to derive the facts that are only implicitly contained by the explicitly encoded facts. Thus querying would be applied to the extended set of facts after the inference step.

In this paper we present our initial results of an investigation into the comparison of the two approaches described above. While our ultimate goal is to use quantitative metrics, such as processing time, bandwidth usage, precision, recall and F-measures to evaluate the two approaches, here we show how we approach this problem and our qualitative assessments of the two approaches based on our initial results.

2. EXPERIMENT SETUP AND PROCESS

In order to make a reasonably fair comparison of the two approaches, one should use the same data sets and the same queries. Such a requirement is not easy to satisfy due to the fact that data sets must first be preprocessed so that they are in the formats appropriate for the processing engines, i.e., for an XQuery processor and for an OWL inference engine. Second, the queries must be translated to the query languages - XQuery and SPARQL. And finally, the XML approach requires a XSD schema, while OWL requires an ontology.

To make this comparison relevant to the efforts of the wireless communications community we ground it in the Model-Based Spectrum Management methodology (MBSM) [7, 8]. The intent of the MBSM approach is to concentrate on the consumption of spectrum, provide computational methods for assessing compatibility among models, serve as a loose coupler for spectrum management systems and enable further extensions of spectrum use and sharing.

The MBSM approach uses Spectrum Consumption Models (SCMs) - data structures that can represent information that is useful for spectrum management. SCMs can serve as specifications of policies for the use of spectrum, representations of the current consumption of spectrum, or requests and authorizations of spectrum use. SCMs are expressed in Spectrum Consumption Modeling Markup Language (SCMML) [7] that is defined by an SCMML XSD schema.

In our previous work [9], we mapped the SCMML schema to Web Ontology Language (OWL) [5]. Then we added additional axioms to the generated OWL representation in order to relate it to other ontologies, e.g., to the Cognitive Radio Ontology (CRO) [10]. The resulting representation - the SCMML ontology (SCMMLO) - was then used to annotate some facts explicitly and to infer implicit facts from the data.

As mentioned above, this work is a continuation of that work. We thus make use of SCMMLO. Additionally, we reuse the scenarios of two spectrum management related use cases described in [9]. For the sake of self-containment of this paper, we briefly describe the two use cases in the next section.

This paper uses examples of queries related to dynamic, opportunistic spectrum access on which the two approaches are compared. The specific queries are discussed in the next section. In general, the queries are about interference that may result from a transmission by a given cognitive radio. In these scenarios, a cognitive radio invokes its reasoner in order to infer whether a specific kind of transmission in a specific situation could cause interference or not. The basis for such an inference is a policy (or policies) that need to be analyzed by the processing components associated with the particular approaches - XQuery processor and OWL based inference engine.

The queries are formulated as part of the development of the use cases. They are then manually mapped to each of the query languages - XQuery and SPARQL. The queries are matched against the policies by two query processing engines. For XML data we use XMLSpy [11], which has a built-in XQuery processor. For the ontology-based approach, we used BaseVISor [12], an inference engine for OWL2 RL (OWL 2 RL is one of the profiles of the OWL language). We used the SPARQL format of queries only in the manual process of query translation, while the final representations of the queries follow the BaseVISor query syntax.

3. USE CASES AND XML-BASED PROCESSING

This section presents how we model two use cases using SCMML and XQuery languages. The first of the use cases considered in this paper is related to the querying of reported transmitter movements (locations) - whether the transmissions from these transmitters located in particular areas are compliant with the policies applicable to the locations. The second one deals with assessing whether signals from specific transmitters to a target receiver might interfere other receivers; this inference must be done before actual transmission occurs. The procedures of representing queries in XQuery language and the results obtained by using a XQuery processor embedded in Altova XMLSpy 2016 XML editor [11] are described.

The construction of the descriptions of the use cases in XML follows the hierarchical structure shown in Figure 1. Layer 1 includes the SCMML XSD schema. This schema is then extended for each of the use cases. The extension process consists of importing (the *import* relation shown in the figure) of the SCMML schema and then adding new concepts that are specific to the use cases. These two XSD schemas are then used to annotate (mark up) the particular pieces of information relevant to the use cases.

3.1. Location Use Case

3.1.1. Scenario Description

In Dynamic Spectrum Access (DSA) the allocation of spectrum to particular radios should be done so that interference to other

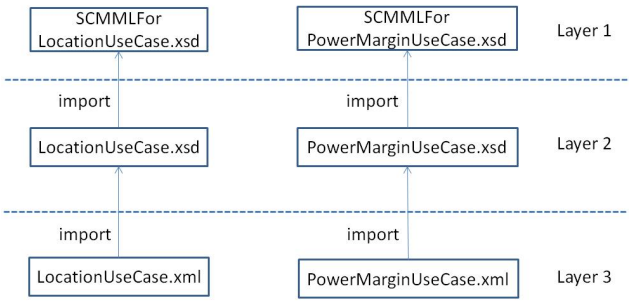

Figure 1: Structure of mapping from OWL to XSD.

radios operating in the same area, e.g., primary users, is avoided. In order to make a decision on which mobile stations can use the particular spectrum, the base station needs to have some information about the mobile station and its needs. For exchanging information about spectrum consumption and needs, the communication nodes can use SCMs described earlier in this paper. An SCM can serve either as a constraint or as an authorization, i.e., it can specify what frequency bands can/cannot be used in what subareas within the transmission scope of a base station. This information can be stored in, or retrieved from, a database of the corresponding base station. Thus the SCM provides information that is useful for making spectrum allocation decisions by the base station. In its inference, the base station makes use of some facts related to spectrum availability in a particular location, e.g., information about the transmitters/emitters located in the area, or definitions of subareas and regulatory policies applicable to the specific subareas. Figure 2 shows a hypothetical scenario discussed in this paper.

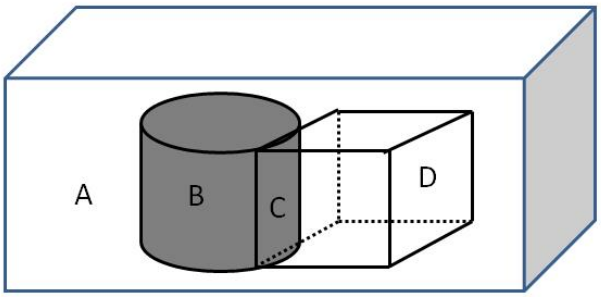

Figure 2: Location use case: Scenario.

Assume that the (rectangular) cuboid in Figure 2 shows the coverage of the base station (base station is not shown here) for this scenario. Suppose that currently four secondary transmitters A, B, C and D, for providers AT&T, T-Mobile, Verizon and Sprint respectively are located in the space covered by the base station. The transmitter information is shown in Table 1 and the policy for spectrum restrictions in each space is shown in Table 2. Both tables are stored in base station database.

As shown in Figure 2, A is outside of both spaces (cylinder and small cuboid), B is within the cylinder, C is in the intersection of the cylinder and the small cuboid, and D is within the small cuboid. Since A is outside of both spaces, according to the policy shown in Table 2, it cannot be assigned any spectrum. Now assume that C moves to a new area. In order to transmit, it sends a request for a permission to transmit to the base station. The request is expressed in the language in which the facts necessary for making a permission decision, in this case it is XML. The base station invokes its dynamic spectrum selection tool (in this case this tool uses XQuery) that executes its policies based on the received request and the data that is kept in its database, derives a decision and provides the spectrum assignment to the mobile station. It is worthwhile mentioning that although the request and the reply are expressed in XML at both the mobile station and the base station, the messages are, or at least can be, compressed before sending and decompressed after receiving so that the communication bandwidth overhead is minimized.

Table 1: Transmitter Information.

Transmitter	Type of Stakeholder	Allocated Spectrum
TransmitterA	AT&T	600-606MHz
TransmitterB	T-Mobile	300-306MHz
TransmitterC	Verizon	600-606MHz
TransmitterD	Sprint	312-318MHz

3.1.2. Modeling the Use Case with SCMML

Although SCMML provides a variety of types that can be used for annotating data, some additional types are necessary to model this use case. For instance, this use case requires to represent such geometrical shapes as Cylinder, which is defined in SCMML.xsd (see Listing 1).

Listing 1: Structure of "Cylinder" in SCMML

```
<xs:complexType name="Cylinder">
  <xs:all>
    <xs:element name="Base" type="Circle"/>
    <xs:element name="Height" type="Distance"/>
  </xs:all>
</xs:complexType>
```

However, SCMML does not provide types to annotate other shapes - cuboid or even cube. Such shapes need to be defined. There are various ways to achieve this. First, we can simply define the new types independently of what is available in SCMML, simply adding more types. While this is not the best option from the point of view of interoperability and reuse, in some cases this is the only option. A better option is to *extend* the types from SCMML by using the "extension" construct of XML. However, the use of this option is limited by the power of this construct. Consequently, we relied on both of these mechanisms for modeling the use cases described in this paper. As an example, Listing 2 shows the definition of cuboid. This kind of extensions are stored in the SCMMLForLocationUseCase.xsd

Table 2: Spectrum restriction policy.

Type of Stakeholder	Location Name	Type of Location	Spectrum	Allow to Use
AT&T	TestCuboid	Cuboid	600-606MHz	N
AT&T	TestCylinder	Cylinder	600-606MHz	N
AT&T	Outside	N/A	600-606MHz	N
T-Mobile	TestCuboid	Cuboid	300-306MHz	N
T-Mobile	TestCylinder	Cylinder	300-306MHz	N
T-Mobile	Outside	N/A	300-306MHz	N
Verizon	TestCuboid	Cuboid	600-606MHz	Y
Verizon	TestCylinder	Cylinder	600-606MHz	Y
Verizon	Outside	N/A	600-606MHz	N
Sprint	TestCuboid	Cuboid	312-318MHz	Y
Sprint	TestCylinder	Cylinder	312-318MHz	N
Sprint	Outside	N/A	312-318MHz	N
...	...	...	...	...

file. To create this file, first we import SCMML.xsd and then insert the additional definitions as in this example.

Listing 2: Structure of "Cuboid"

```
<xs:complexType name="Cuboid">
 <xs:all>
  <xs:element name="id" type="xs:string"/>
   <xs:element name="base" type="Rectangle"/>
   <xs:element name="height" type="scmml:Distance"/>
 </xs:all>
</xs:complexType>

<xs:complexType name="Rectangle">
 <xs:all>
  <xs:element name="NE_Vertex" type="scmml:Point"/>
  <xs:element name="SW_Vertex" type="scmml:Point"/>
 </xs:all>
</xs:complexType>
```

3.1.3. *Representing and Processing Queries in XQuery*

The processing of XML data consists of three steps:

1. *Represent transmission policies in XML.* In this case, the policies shown in Table 2 need to be represented (see Listing 3 for an example). First, a new type "Policy" is added to the XSD definitions. Then this type is used to represent the specific instances of the Policy type and added to the XML data representation file.

2. *Annotate the data in XML.* For this particular case, the data consists of the facts about the communication environment, including transmitters and receivers that operate in the given area (e.g., their locations, transmit power) as well as the specific request for transmission.

3. *Execute a query.* In this case the query is about whether the transmission policy referred to in point 1 above allows to transmit as specified in the request referred to in point 2.

Listing 3: An example of Policy

```
<xs:complexType name="Policy">
 <xs:all>
  <xs:element name="allowedStakeholder" type="xs:string
  <xs:element name="appliesToRegion" type="Shape"/>
 </xs:all>
 </xs:complexType>

<xs:complexType name="Shape">
 <xs:choice>
    <xs:element ref="shapes:Cuboid"/>
    <xs:element ref="scmml:Cylinder"/>
  </xs:choice>
</xs:complexType>
```

The main task of the processing of the query for this use case is to infer whether the transmitter is located inside of the cylinder and to calculate whether the horizontal distance between the point where the transmitter is located and the center of the circular base of the cylinder exceeds the radius of the base. For points A and B having longitude values $LonA$, $LonB$ and latitude values $LatA$, $LatB$ (in radians), and R being the radius of the Earth (in meters), the horizontal distance (HD) is calculated by the following formula. All of the operations needed to define such a computation are part of the functional libraries supported by XQuery.

$$HD = R \cdot arccos(cos(LatA) \cdot cos(LatB) \cdot cos(LonA - LonB) + sin(LatA) \cdot sin(LatB))$$

To express queries, we use XQuery FLWOR expressions [4] to derive all the results about which transmitters can transmit using their current allocated spectrum within the shapes used in the policies for the region covered by the use case. The query is (partially) represented in Figure 3.

CHECK-POLICIES(P, T)

1 **for** each policy $p \in P$
2 **for** each transmitter $t \in T$
3 **if** $p.allowedStakeholder == t.stakeholder$ and $t.location \in p.region$
4 PRINT-ALLOWED$(t, p.region)$

Figure 3: A query expressed in FLWOR

3.2. Power Margin Use Case

3.2.1. Scenario Description

When two radios communicate via a channel shared by other nodes, a multiple access protocol must be utilized. Contention protocols resolve a collision after it occurs, while collision-free protocols ensure no collision. However, the former requires a lot of back and forth communication to deal with collisions while the latter reduces channel utilization ratio when transmission overload is not high. Therefore, it is necessary for a transmitter to detect whether a signal may interfere with receivers that are not target receivers before actual transmission occurs. Here we discuss a Power Margin use case to explain how to use the XML-based approach to address these issues. In this use case, a request is required from the transmitter to the base station on whether transmittion is allowed, before transmitting. The base station then returns a decision derived by its query engine. Figure 4 shows a hypothetical scenario.

As shown in this figure, this scenario includes four transmitters, marked as TA, TB, TC and TD, respectively. Additionally four receivers are marked as RA, RB, RC and RD, respectively. Suppose TA is transmitting to RA and TB is transmitting to RB. If TC wants to transmit to RC and TD to RD at the same time, we want to know whether these transmissions may interfere with the other receivers that are not target receivers for these transmitters.

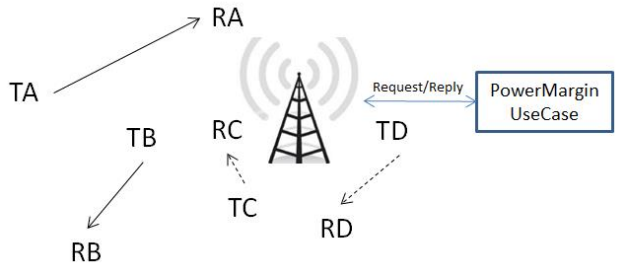

Figure 4: Power Margin use case scenario.

3.2.2. Relevant MBSM Concepts

Several concepts are proposed in the MBSM approach to deal with this problem [7].

Spectrum mask: Spectrum mask specifies the power-spectrum density vs. frequency, relative to the total power of a transmitter. In the scenario considered here, power spectrum density varies by location since the signal attenuates with propagation and varies with terrain.

Underlay mask: Underlay mask specifies the power spectrum density of a signal that a receiver can tolerate from a remote interfering transmitter; it is represented as a function of relative power versus frequency. It defines the maximum power level of the anticipated interference at the receiver as a function of frequency.

Power margin: Power margin is the minimum power adjustment that would have to be made to make the spectrum power mask and the underlay mask meet. When the masks are represented by inflection points, power margin is the adjustment that would have to be made across the overlapping inflection points that would cause the two masks meet.

Both spectrum mask and underlay mask are usually represented as piecewise linear graphs of power spectrum density. Figures 5 to 8 show spectrum masks of transmitters TA, TB, TC and TD at the the locations of receivers RA, RB, RC and RD, respectively, the underlay masks of receivers RA, RB, RC and RD, respectively, as well as the power margins.

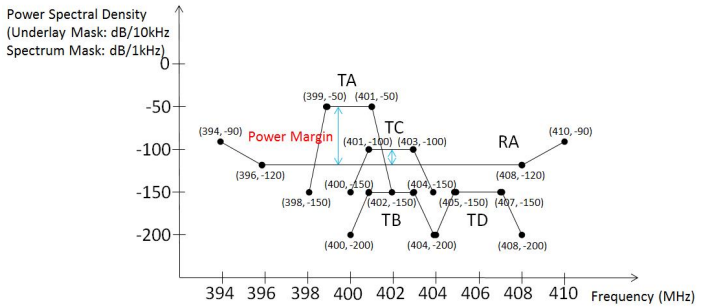

Figure 5: Spectrum masks vs. the underlay mask of RA.

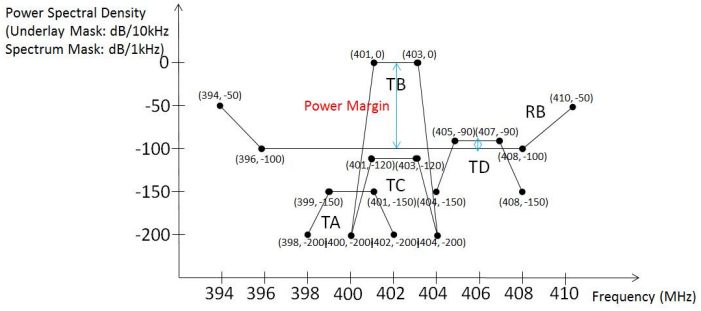

Figure 6: Spectrum masks vs. the underlay mask of RB.

Power margin can be computed using the total power method or the maximum power spectrum density method. In our implementation of this use case we used the latter method. This method is used when multiple interferers do not coordinate their

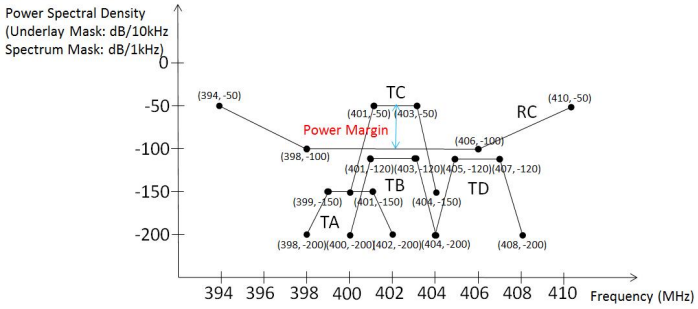

Figure 7: Spectrum masks vs. the underlay mask of RC.

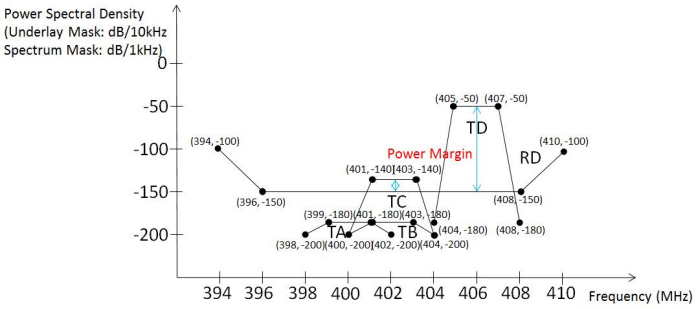

Figure 8: Spectrum masks vs. the underlay mask of RD.

interference with each other (as was assumed in our use case) to ensure that they collectively stay within the limits of allowed interference. For this, each of the transmitters must stay within the limits of the underlay mask.

In this method, first, both the underlay mask and the interfering signal's spectrum mask are converted to the same resolution bandwidth. For instance, in our scenario the bandwidth of the spectrum mask and the underlay mask were 1kHz and 10kHz, respectively. Second, we had to infer the value of the power margin.

Following this procedure, we can estimate that TC would interfere RA (seen from Figure 5) and RD (seen from Figure 8), while TD would interfere RB (seen from Figure 6), i.e., both transmitters TC and TD should not transmit at the same time to avoid interfering the receivers that are not their targets.

3.2.3. Modeling the Use Case with SCMML

We use the same approach to model the use case with SCMML as for the Location use case. Only one new type is added to SCMML (the Segment complex type).

3.2.4. Represenation of Queries in XQuery

As discussed before, the solution to this problem requires querying whether sufficient power margins exist for the receiver-transmitter pairs. Again, we use XQuery FLOWR expressions to get the final query results. The XQuery query expressions for the queries for this use case look very much like the queries for the Location use case. Running the queries using an XQuery processor derived correct conclusions regarding the interference of the potential transmissions to the receivers that would not be the targets for the transmissions.

4. USING OWL VS. XSD

The main objective of this paper is to explain how a purely XML-based approach to the MBSM can be used. In this section we provide some observations about the comparison of this approach vs. the approach based on OWL, as presented in [9]. These two approaches were implemented for the same two use cases, which allows us to make some reasonable comparisons. Moreover, similar processing steps, as described in Section 3.1.3., were used. The differences between these two approaches are as follows:

1. Using ontologies (CRO) vs. XSD schemas (SCMML) for capturing the generic structures of knowledge representation.
2. Representation of policies in OWL vs. XML plus XSD.
3. Representation of instances: using OWL vs. XML.
4. Representation (and execution) of queries in SPARQL vs. XQuery.

4.1. Mapping to OWL vs. XSD

During the process of mapping two use cases to OWL and XSD, some of the features that OWL supports may not be applicable or convenient to use in XSD. These features are listed as follows:

Amount of hard-coding required: OWL uses the "property" concept to represent relations between instances of classes. Properties may have a domain and a range specified. Properties link individuals from the domain to individuals from the range [13]. However, properties don't necessarily need to be restricted to the specific classes specified as their domains and ranges. They can be used to link other classes as long as the restrictions specified in the ontology are not violated. XSD, however, is not flexible enough to deal with this case. One possible solution is that XSD developers have to hardcode all the possible relationships associated with the complex types being defined. Another way is to define the complex type extensible with the tag <xs:any/>. Otherwise, the XML complier will display an error demonstrating inconsistency between XML data and its schema.

Another benefit of mapping to OWL is that any component can be easily identified via IRI [14]. Therefore, multiple instances of the same type can be easily distinguished by their IRIs. In XSD, however, it is not that simple. All the complex types and associated XML data are expressed as elements. If we don't attach a label to each element, it is impossible to distinguish or mark each element with the same type.

Another aspect of inflexibility of modeling in the XML approach lies in the connection between class/type and their individuals/XML data. OWL can easily connect individuals with

their affiliated class via axioms rdf:type. In XML, however, users have to develop a schema that contains all the types that are used in XML data. Then XML data are organized in a structure defined in the schema. In this way, users have to spend more time developing the schema for XML data. Moreover, users have to modify the schema every time when new types are used in XML data or some inconsistency occurred due to the insertion of new XML data.

Expressiveness: All the elements in XML are organized in a tree structure, while in OWL all the classes are connected via properties, thus forming a graph. Since graphs are generalization of trees, the net result is that OWL is more expressive than pure XML (although there are some cumbersome ways to represent graphs in XML). Since tree structure doesn't contain loops, it is rather difficult to describe relations that form circuits in XSD. Even though XSD supports element and attribute groups, which can be referred to via their group names, it doesn't contain reference to single elements. Theoretically, infinite nested structure is required to describe relations with circuits. OWL, however, has the capability to describe various types of relation. For instance, reverse properties pairs can be used to interconnect two classes.

Support of inheritance: XSD supports signal inheritance relationship among different types via <xs:extension base="XXX"/> tag. However, it doesn't support multiple inheritance. XSD is not able to handle the case in which a type inherits from multiple ancestors. OWL can easily express multiple inheritance relationships via the axioms that use rdfs:subClassOf. For instance, ":A rdfs:subClassOf :B, :C" declares that component A is sublcass of component B and C.

Support of polymorphism: The main benefit of OWL with respect to supporting polymorphism is primarily due to its automatic inference capability. The reason why we list it here separately is just to highlight its importance. In OWL, if we know the fact that "LocationUseCase:TestCuboid rdf:type LocationUseCase:Cuboid, LocationUseCase:Cuboid rdfs:subclassOf LocationUseCase:Polyhedron", the inference engine will infer that "LocationUseCase:TestCuboid rdf:type LocationUseCase:Polyhedron". In XSD, however, if an element is defined as Polyhedron complex type, and if the XML data contains an element with the complex type Cuboid, the error will be flagged by the XML validator. This is because the XML validator is not able to recognize it as an element of Polyhedron complex type. The same problem exists for properties. Such a feature dramatically decreases the flexibility and extensibility of modeling with pure XML.

Automatic inference: The biggest difference between XSD and OWL lies in the power of making automatic inference. OWL provides various kinds of useful features to model relationships among components. OWL inference engine is then is able to generate more implicit facts from such data. XSD, unfortunately, is not able to generate any inferences. XML data has to be either hard-coded into the XML data file, or all the inference steps need to be hard-coded into the XQuery expressions (see discussion below).

Expression on intermediate variables: XQuery supports path expressions that are made of one or more steps that are separated by a slash(/) or double slashes (//), which makes it much easier to store intermediate variables. In BaseVISor queries or SPARQL, we have to write out all the subpaths from the source to the target. For example, if we want to get the altitude value of the circle center of a cylinder in XQuery, it can be expressed as follows:

```
$altitudeForCircleCenter :=
   $cylinder/scmml:Base/scmml:Center/scmml:Altitude
```

In OWL, however, we have to use nested tags in order to retrieve the values, which is shown as follows:

```
<Individual variable="AShape" rdf:type="scmml:Cylinder"
 <scmml:hasBase variable="BaseForACylinder">
   <scmml:hasCenter variable="CircleCenter">
    <scmml:hasAltitude variable=
    "AltitudeForCircleCenter">
     <scmml:hasValue variable=
     "AltitudeValueForCircleCenter"/>
    </scmml:hasAltitude>
   </scmml:hasCenter>
 </scmml:hasBase>
</Individual>
```

Support of function library: XQuery has a better support for arithmetic operations. It contains a math function library with prefix 'math:', for example, math:cos($arg1 as xs:double?) as xs:double? expression returns the cosine of the argument, expressed in radians. BaseVISor semantic queries, however, have to use nested tags to express function names and arguments, which makes the arithmetic operations very verbose. In addition to the support of arithmetic operations, similar as imperative languages, XQuery provides various kinds of functions that deal with strings, dates, times, etc. [15]. Additionally, users can develop own functions in XQuery, if necessary.

5. CONCLUSION AND FUTURE WORK

This paper discusses the continuation of our previous work that deals with the conversion of SCMML to OWL and rules and the use of such representations in the process of spectrum management by proposing two use cases as examples. This paper reuses two use cases and uses a pure XML approach to model them using SCMML. The XQuery language is used to represent the queries. The implemented queries demonstrate that it is possible to use pure XML and XQuery as a non-semantic approach to derive desirable conclusions. The main point of this paper is to qualitatively compare OWL and XML approaches with respect to modeling upon SCMs expressed in SCMML and associated queries. Our conclusion from this preliminary comparison is that OWL is a better tool for modeling, especially in the aspect of the amount of hard-coding required, expressiveness, support of inheritance and polymorphism and automatic inference. XML approach, on the other hand, has some advantages in querying data. The query expressions in XQuery will be more succinct due to the strong support by the XQuery function library. Note

however, that XQuery does not provide any semantic inference rules, and thus inferences must be explicitly hard-coded into the queries. Thus while the XQuery processing is simple to query existing data, the processing of inference is very difficult, just like for any procedural code. The main reason for this is that OWL has model theory based semantics that the inference engines implement. The inferences derived by OWL reasoners are thus guaranteed to be *sound* (i.e., the derived facts logically follow from the initial facts). This feature is very difficult to achieve in XQuery.

The next step will be to compare these two approaches in a quantitative way. We plan to use various metrics for this purpose. The metrics will include precision/recall of query results, bandwidth usage, the size of encoding of instance data, query encoding size, inference and query processing times, query complexity. Additionally, dynamic behaviors such as supporting new devices and funtionalities will also be taken into consideration. Moreover, more use cases with different types and parameters will also be added to test data sets to make the evaluation results more reliable and convincible .

We also plan to provide our results to the efforts of developing a standard language for describing policies. This kind of language is being standardized by the IEEE 1900.5 Working Group. The work of this group has resulted in the publication of the requirements for such a policy language [16]. Currently, this Working Group is preparing a standard (IEEE 1900.5.1) for a policy language that partially satisfies the requirements specified in [16]. Additionally, this group is finalizing another related standard (IEEE 1900.5.2), which will capture the specification of Spectrum Consumption Models [17].

REFERENCES

[1] in *Cognitive Radio Technology*, B. Fette, Ed. Academic Press, Elsevier, 2009.

[2] M. M. Kokar and L. Lechowicz, "Language issues for cognitive radio," *The Proceedings of the IEEE*, vol. 97, No. 4, pp. 689–707, 2009.

[3] W3C, "XML Schema," 2015, `http://www.w3.org/XML/Schema`.

[4] W3C, "XQuery 3.0: An XML Query Language," 2014, `http://www.w3.org/TR/xquery-30/`.

[5] W3C, "OWL 2 Web Ontology Language Document Overview," 2009, `http://www.w3.org/TR/owl2-overview/`.

[6] T. Gruber, "Ontology," in *The Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer, 2009, pp. 1963–1965.

[7] J. Stine and S. Schmitz, "Model based spectrum management," The MITRE Corporation, Tech. Rep., 2013.

[8] L. Grande, M. Sherman, H. Zhu, M. M. Kokar, and J. Stine, "IEEE DySPAN 1900.5 Efforts To support Spectrum Access Standardization," in *2013 IEEE Military Communications Conference*. IEEE, 2013, pp. 1750–1755.

[9] Y. Chen, M. M. Kokar, J. Moskal, and D. Suresh, "Mapping spectrum consumption models to cognitive radio ontology for automatic inference," in *Wireless Innovation Forum Conference on Wireless Communications Technologies and Software Defined Radio*. Wireless Innovation Forum, 2015.

[10] WinnF, "Description of cognitive radio ontology," Wireless Innovation Forum MLM Working Group, Tech. Rep. WINNF-10-S-0007, 2010, `http://groups.winnforum.org/d/do/3370`.

[11] "Altova XMLSpy 2016," 2015, Altova, Inc:`http://www.altova.com/products.html`.

[12] C. Matheus, M. M. Kokar, and R. Dionne, "A demonstration of formal policy reasoning using an extended version of BaseVISor," in *IEEE Workshop on Policies for Distributed Systems and Networks, POLICY 2008*, 2008.

[13] M. Horridge, "A practical guide to building owl ontologies using protege 4 and co-ode tools edition 1.3," The University Of Manchester, Tech. Rep., 2011.

[14] M. Duerst and M. Suignard, "Internationalized resource identifiers (IRIs)," 2005, `http://tools.ietf.org/html/rfc3987`.

[15] W3C, "XPath and XQuery Functions and Operators 3.0," 2014, `http://www.w3.org/TR/xpath-functions-30/`.

[16] "IEEE Standard for Policy Language Requirements and System Architectures for Dynamic Spectrum Access Systems. IEEE Std 1900.5^{TM}-2011," IEEE Communications Society, 2011.

[17] IEEE, "IEEE DySPAN 1900.5.2 standard," 2015, `http://grouper.ieee.org/groups/dyspan/5/index.htm`.