Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

PROTOTYPING SCA TRANSCERIVER APIs USING A GENERIC REASONER API

Durga Suresh (Northeastern University: Boston, MA; suresh.d@husky.neu.edu); Mieczyslaw Kokar (Northeastern University: Boston, MA; m.kokar@neu.edu); and Jakub Moskal (VIStology, Inc.: Framingham, MA; jmoskal@vistology.com)

ABSTRACT

API's are developed for different protocol layers, each with a specific purpose and particular hardware and software needs. Within the realm of the SDR there are many different APIs that are associated with transmitters, receivers, specific purpose applications for military operations or general research. These APIs are then implemented within a common SCA architecture, leading to a great advantage of interoperability among various radios and being platform independent. The standard practice of developing an API for an SDR is by first describing it in UML. While UML tools provide some methods for syntactically constraining the development of a specification of a system, they don't support the capability of verifying or enforcing the semantic constraints. Consequently, the semantic interpretation of the constraints imposed by an API is done by humans. This paper discusses the potential uses of languages with formal semantics (e.g., OWL), in addition to UML, in the development of the SDR API's. In particular, it investigates the use of the concepts from the cognitive radio ontology (CRO) to express a Transceiver API and then using a reasoner to analyze the API specification, e.g., checking its logical consistency and querying. This paper proposes that for the purpose of analyzing the specifications of API's, instead of implementing each individual API in a programming language, the CRO could be used to formalize the API and then an implementation of a generic, ontology-bound API be used. The user will thus not have to implement a new API in a programming language, but instead use this generic API for analysis and partial testing. However, the API might need to be eventually implemented, e.g., in case the generic API does not provide the sufficient efficiency in terms of computational complexity.

1. INTRODUCTION

Ontology defines basic terms and the relationships between these terms [15]. It can be used to share information between people and machines. It can further help us define the domain specific knowledge. In the Cognitive Radio (CR) domain, ontologies were proposed to enable Software Defined Radio's (SDR) to achieve interoperability by exchanging knowledge regarding communication parameters and protocols [7].

In current practice API's are found throughout a system at all levels where modules or components interface with the hardware and software of a SDR as shown in Figure 1. The Transceiver API is one such API that uses CORBA as middleware. The API's within the realm of the cognitive radio and the SCA are specified using UML and are then implemented on a need basis. The problem with this scenario is that in order to analyze a new proposed API, one has to implement it in a programming language. Moreover, since there are so many different API's, it is not easy to analyze the relationships between the different API's. .

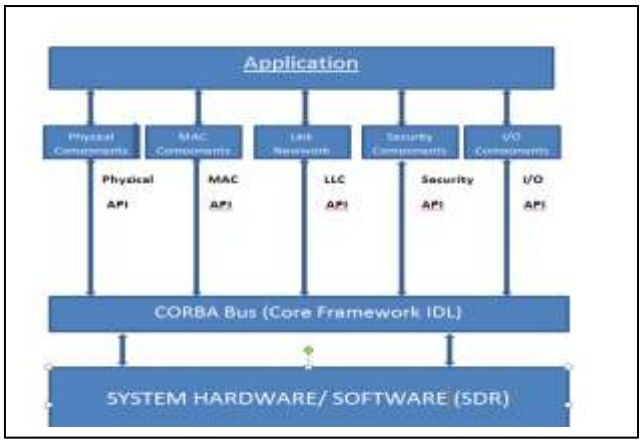

Figure 1: Current Interaction of API to SDR

The idea proposed in this paper is to complement the UML capabilities by taking advantage of the existing generic inference tools for the purpose of analysis of new API's. More specifically, the idea is to take the specification of an API that is described in UML and map it to OWL (Figure 2). By mapping the API's UML specification to OWL, an application can access the system via the ontology API, instead of via each API individually (as shown in Figure 1). This will provide the analysis capabilities, e.g., (1) API specifications can be checked for consistency and (2) the API's can be queried via a standard query language.

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

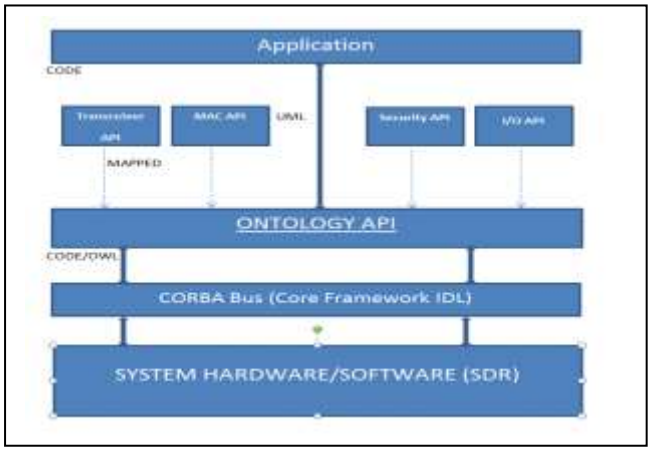

Figure 2: Ontology Based API Design

We investigate the feasibility of this approach by expressing (partially) the transceiver API in OWL and then using a reasoner to analyze the specification for consistency and for querying. The ultimate idea is to extend this API specification mapping to include all API's so that we do not use a plethora of API's but use one ontology in which all the API's are expressed. Once the API specification is mapped and the consistency checked, querying the API specification can be done using SPARQL.

Classically the specifications were written using the Unified Modeling Language (UML). However, other languages could be used, too, e.g., the Specification and Description Language (SDL). The SDL standard as described by the International Telecommunication Union's Z.100 [13] document describes the use of SDL as an unambiguous specification and description of behavior of telecommunication systems. The use of SDL has been known in the field of telecommunications for some time now, including the application to network management, communication protocols and telecommunication services. SDL, when used in combination with Message Sequence Charts (MSC), can be used to test system descriptions.

The paper is organized as follows. Section 2 is a literature review of the API's that exist for the SDR and the description of the transceiver API. Section 3 discusses the mapping of the transceiver API UML specification to OWL. Section 3 also shows how such a specification is checked for consistency and how it can be queried. Section 4 gives the conclusion and future work to be done.

2. LITERATURE REVIEW

The development of API's for the SDR has been an important charge for the SCA community. The needs and benefits of an API are twofold: 1) application portability and 2) ease of upgrade/enhancement [4]. The number of API's that are used by the Joint Tactical Radio System (JTRS) is at least twenty, where there is a specific API for each functionality associated with software radios. The JTRS infrastructure has 1) primitive APIs that provide messaging and signaling interfaces and 2) complex APIs that define radio devices and services. The primitive APIs include Device IO API, Device Packet API, Device Simple API and Device Message APIs. The complex APIs include Ethernet Device API, Serial Port API, Audio Port API, Vocoder Service API and Modem Hardware Abstraction Layer (MHAL) API [5]. There are also new API's developed every day that add to this list of APIs that are used for communication, security and protocols. This paper provides a way to consolidate the different API's by mapping them to the CRO. The CRO then is used, involving a reasoner, to issue queries regarding specific methods from the APIs and to analyze the API specification.

The Transceiver API is the result of work of the Wireless Innovation Forum's Transceiver Subsystem Interface Task Force that defines the "Transceiver Facility". The term "transceiver" is used to encapsulate the entire set of hardware and software components within a radio set necessary to convert a low-power RF signal to digital baseband on the receiver side, and reciprocally to convert digital baseband signals to low-power RF on the transmit side [8]. The transceiver subsystem derives from the contraction of "transmitter/receiver"; it is a part of a radio chain that transposes for transmission, baseband into radio signal and for reception, radio signal to baseband [5]. The transceiver subsystem is considered a part of the physical layer and is comprised of the modem and the antenna subsystem. The rationale for using such a standard based specification is to increase interoperability between the Waveform Applications and Transceiver Subsystem [5].

The formal representation of information and knowledge is becoming a common practice in software engineering. The paradigms that have emerged to support this representation are 1) modeling and 2) ontologies. Model Based Development (MBD) begun with OMG's UML [10]. Ontologies were developed by the artificial intelligence community in the W3C's language OWL [11]. UML, SDL and MSC's can be used for modeling and representing communication services, however recently UML has dominated this field. An SDL specification can be used to define system behavior and can be seen as a sequence of responses to given stimuli [13]. SDL semantics is described in the Z.100 specification but the semantics is not machine readable in the sense that there are no formal representation of the semantics that is interpreted by any inference engine. Although the SDL semantics presented in the SDL Z.100 document is precise, the semantics is written in natural language (text) and thus only human-interpretable. To the best of our knowledge, a generic inference engine does not exist that would deduce any implicit information from an SDL specification.

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

This paper talks about mapping UML to OWL and compares the use of these two languages for analyzing software defined radio APIs. It is important to compare both syntax and semantics of the languages. Syntax can be further classified into abstract syntax and concrete syntax and semantics can be further classified into semantic domain and semantic mapping [12]. A language comparison should take all of these into consideration. The most important issue in language comparison is subject and expressiveness.

As part of an OMG's effort, ODM has been developed as a bridge for mapping UML to OWL [2]. In our mapping of API specifications from UML to OWL, ODM plays a very significant role. Note, however, that ODM provides only a partial mapping, not a complete mapping. The main reason for this partiality is that some of the features of these two languages are very difficult, if not impossible, to reconcile. This issue, however, is outside of the scope of this paper.

3. METHODOLOGY

3.1 Mapping UML to OWL

Unified Modeling Language (UML) is a language that is used to model application structure, behavior, architecture and is also used for representing business processes and data structures [6]. It is a standardized general-purpose modeling language that is controlled by the OMG. Primarily it is used to create visual models. UML combines data modeling, business modeling, component modeling and object modeling.

Web Ontology Language (OWL) is an expressive language for representing and sharing ontologies over the web [9]. It is designed to be used by applications that process content rather than present information. It facilitates greater machine interoperability than other languages by providing vocabulary with formal semantics.

Object Data Metamodel (ODM) is a standard from Object Management Group (OMG) that supports ontology development and conceptual modeling. In provides a framework for ontology creation based on the MOF (Meta Object Facility) and UML. It thus offers a set of metamodels and mappings for bridging the worlds of metamodels and ontologies. ODM defines five metamodels and two UML profiles and a set of QVT (Query, Verify, and Transform) mappings; these are used to map between UML and OWL.

There are many common features between UML and OWL and thus the ODM incorporates this into the mappings. Consequently, UML classes are mapped to OWL classes, UML instances are mapped to OWL individuals (where the OWL individual is independent of the class), UML *ownedAttribute* and *binaryAssociation* are mapped to OWL properties, where the properties can be either local or global. UML subclass and generalizations are mapped to OWL *subClassOf* and *subProperty*. A summary of the mappings is shown below.

Table 1: UML to OWL mapping

UML Elements	OWL Elements
Class, property owned attribute, type	Class
Instance	Individual
ownedAttribute, binary association	Property
Subclass, generalization	subClassOf, subProperty
N-ary association, association class	Class, Property
Enumeration	oneOf
Disjoint, cover	disjointWith, unionOf
Multiplicity	minCardinality maxCardinality
Package	Ontology
Dependency	Reserved name rdf:Property

The table shown above shows the ODM specifications of the mapping between UML and OWL. All of the UML features considered in the scope of ODM have some satisfactory OWL equivalents. Some UML features that have no OWL equivalents are *navigable*, *non-navigable*, *derived*, abstract classifier and classes as instances. Some OWL features that have no UML equivalents are `Thing`, global properties, autonomous individual, `allValuesFrom`, `someValuesFrom`, Symmetric Property, Transitive Property, classes as instances, `disjointWith` and `complementOf`.

Ontology is a conceptual model and the UML diagrams are a rich representation system. UML is used widely and is well supported. Some of the reasons we do not use UML for representing ontologies are: 1) UML does not have a construct for set complement, 2) set intersection cannot be modeled using UML, 3) lack of computer processable semantics makes it impossible to use a formal reasoner with UML.

3.2 Mapping UML Specifications to the CRO

In our work we use the Cognitive Radio Ontology (CRO) developed by the MLM Work Group of the Wireless Innovation Forum [14]. In particular, the CRO is used to represent the UML Transceiver API in OWL (Web

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

Ontology Language) [7]. The CRO at its base uses DOLCE, the Descriptive Ontology for Linguistic And Cognitive Engineering [3]. The DOLCE concepts of Endurant, Perdurant and Quality are known as Object, Process and Attirbute in the CRO. An Object refers to an entity that is wholly represented in one snapshot and Process is partially represented at any given snapshot of time. Attributes are basic entities that can be perceived or measured. The structure of the CRO is based on a principle that the attributes cannot exist on their own; instead they are always associated with a process or an object.

The transceiver API has been used as an example API that is specified in UML and is then mapped to the CRO. The UML diagram for the API specifies classes and parameters that can be adjusted or invoked. When we map the API to the CRO, the methods are mapped as classes and the parameters that are used to describe the class are mapped as method. Figure 3 shows the Transmit Channel related interfaces. It shows the relationship between the different classes and the methods in UML. The transceiver subsystem that is shown is comprised of the Transmitter Channel and the Receiver Channel. The Waveform Application is a composition of software modules which provide a software defined dimension that is essential to the radio capability implementation of the system. The Transceiver Subsystem and the Waveform Application have interfaces between them as shown in Figure 3. They are the Transmit Control and the Transmit Data Push classes. Transmit Control is the interface used to control the Transmit Channel behavior according the Waveform needs. The operations included in the Transmit Control are createTransmitCycleProfile(), configureTramsnitCycleProfile() and setTransmitTime(). Transmit Data Push is the interface used for transferring baseband samples from the Waveform Application to the Transceiver Subsystem. The interface includes the operation pushBBSamplesTx(). The transmit channel shown here is a part of the Transceiver Subsystem and is comprised of four methods: createTransmitCycleProfile(), configureTransmitCycle(), setTransmitStopTime() and pushBBSamples(). When mapping this UML class diagram to the CRO, the subsystems are mapped to objects and the interfaces are mapped to processes. The relationship between the objects and the methods are established using properties.

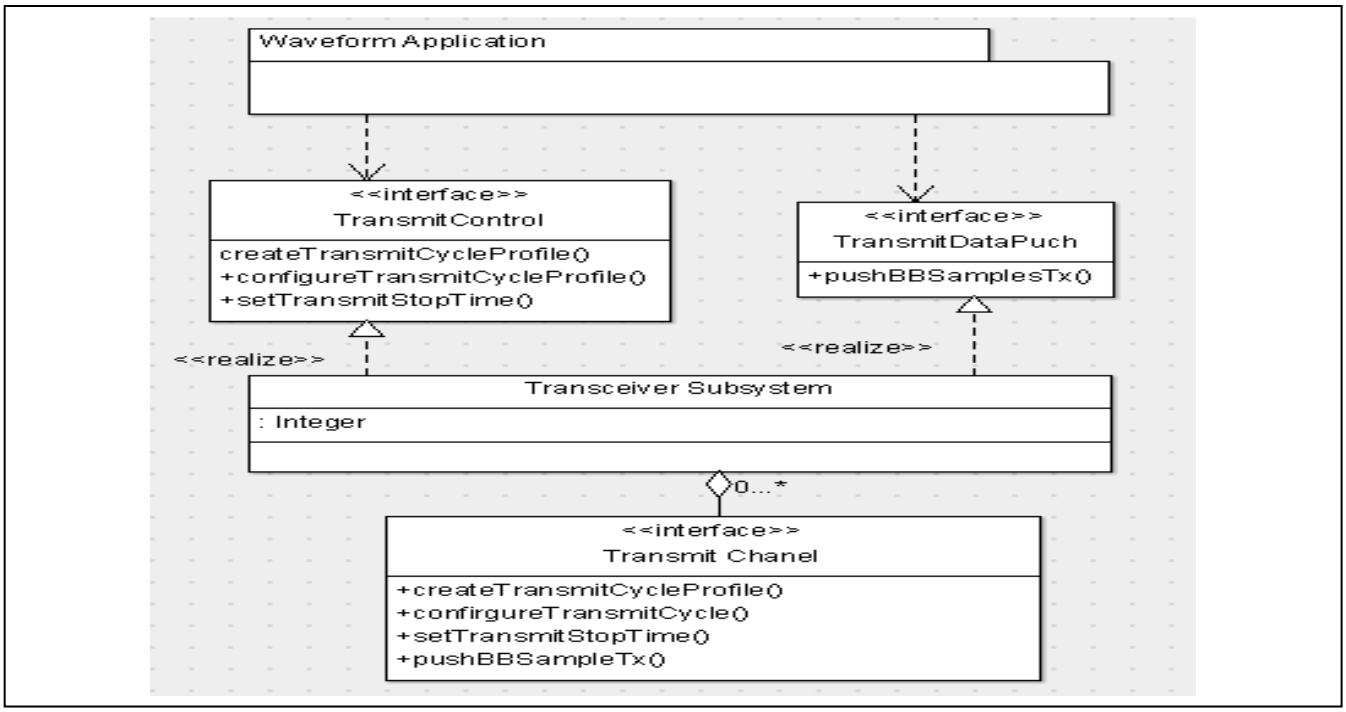

Figure 3: Transmit Chanel Related Interfaces

Figure 3 shows the UML of the Transmit Channel invoked interfaces that have been mapped to the CRO. Here we see that we have a package, two interfaces and two classes that need to be mapped in the CRO.

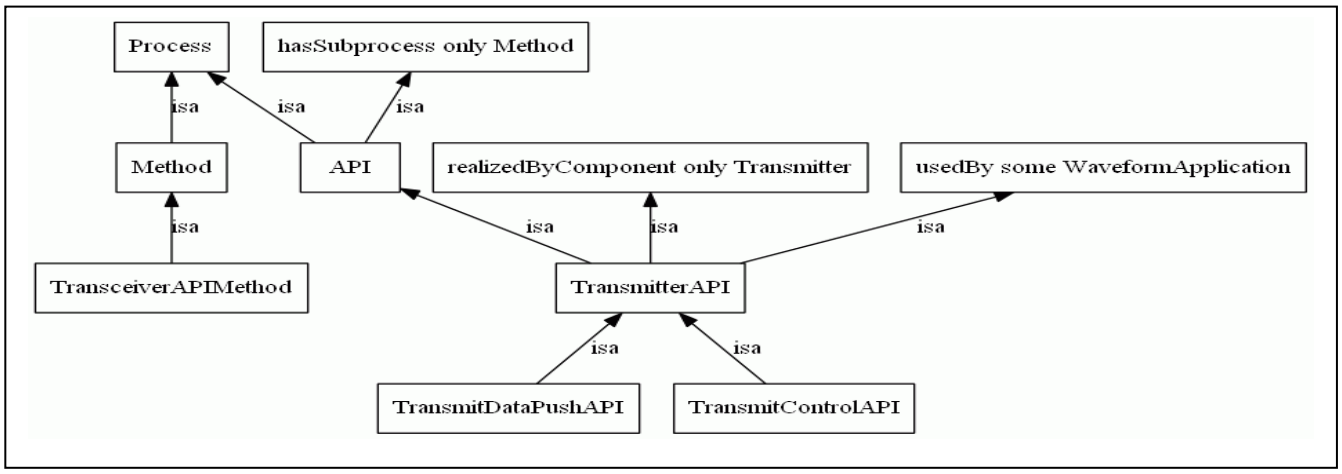

Figure 4: UML mapped to the CRO

Table 2: Transceiver API's UML to OWL mapping

UML Elements	OWL Elements
Waveform Application	Thing -> Object -> Component -> Waveform Application
Transmit Control	Thing -> Process -> API -> TransmitControlAPI
Transmit DataPush	Thing -> Process -> API -> TransmitDataPushAPI
Transceiver Subsystem	Thing -> Object -> Component -> Transceiver
Transmit Channel	Thing -> Object -> Component -> Transmit Channel

The transceiver subsystem is mapped as objects and processes in the CRO. In OWL, there is a universal class `Thing`, whose extent is all individuals in a given OWL model, and all classes are subclasses of `Thing`. Relationships among classes in OWL are called properties. The table below shows the properties that are mapped to OWL from the associations in UML for the Transceiver API.

Table 3: Transceiver API's UML to OWL mapping

UML Association	OWL Property
Dependency	implementsAPI
Association	useAPI
Consistsof	usedBy

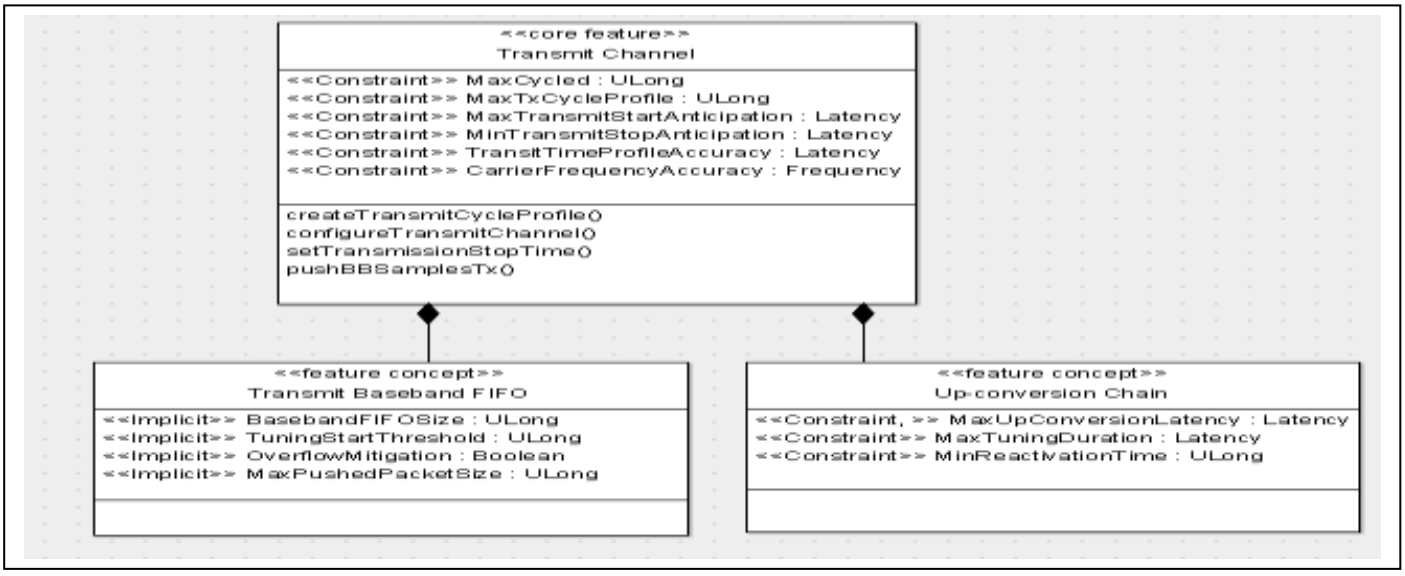

Figure 5: Transmit Channel main composition

3.3 Checking for Consistency

One of the advantages of using OWL over UML is the ability to check for consistency of the specification. For an example of consistency checking, first consider Figure 5 which shows the UML class diagram of the Transmit Channel in the Transceiver API and the classes Transmit Baseband FIFO and Up conversion Chain. This diagram is one of the UML diagrams that were described in the Transceiver API specification.

The solid diamond between the Transmit Baseband FIFO and Transmit Channel denotes composition: a strong form of aggregation where the life time of the component instances is controlled by the aggregate. That is, the parts don't exist on their own ("the whole controls/destroys the parts"). When this relationship is mapped to the CRO, a cardinality relationship is set where the instances of classes Transmit Baseband FIFO and Up-conversion Chain are set as sub-components of Transmit Channel and can have one and only one of each of the instances. To show how this constraint is checked for in OWL, two instances of the class Transmit Channel is created, say T1 and T2, and one instance of Transmit Baseband FIFO is created, say TBF, as shown in Figure 5. TBF is set as sub component of both T1 and T2. According to the cardinality restriction (shown in Figure 6), the Transmit Baseband FIFO can be the sub component of one and only one Transmit Channel.

There is now an inconsistency identified when we use a semantic reasoner. A semantic reasoner is software that is able to infer logical consequences from a set of asserted facts. The reasoner used with our Ontology tool (Protégé 4) is Pellet. Pellet is a Java OWL 2 reasoner that provides reasoning services for OWL ontologies. When the semantic reasoner is run for the scenario presented in the preceding section, we get an error that is shown in Figure 7. The error indicates that this is an inconsistent ontology. The reason for inconsistency is shown as the violation of the constraint that individual TBF cannot have more than one value for the property isSubComp (which violates the cardinality restriction).

The identification of these inconsistencies can be done automatically in OWL rather than searching for them "manually" in a UML class diagram. Implementation of these rules can be easily overlooked in specifications, especially when the UML models are complex.

Figure 6: Cardinality Restriction

Similarly, inconsistencies can be present in the dependency relationships, inheritance relationships, composedOf relationships that exist among the classes, and many others.

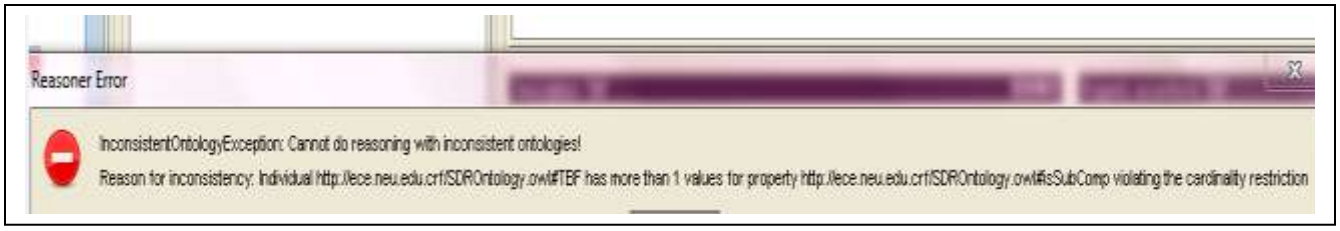

Figure 7: Reasoner Error

Automatic catching of errors by a reasoner can help us identify inconsistencies in the specification and thus let us avoid unnecessary implementation efforts.

3.4 Querying the CRO

Once the consistency of the API mapping is checked, one can then query the specification of the API in the CRO. The fact that a specification is consistent does not imply it is correct. One of the ways to test for the correctness of a specification is to check whether it satisfies various requirements, or in other words, whether it has some desirable (or undesirable) features. If the specification is expressed in a formal language, checking for features can be achieved via querying. SPARQL is used as the query language for ontologies. It is a query language able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. One of the advantages of SPARQL is that it allows for the writing of unambiguous queries. The results are displayed in a table format.

In order to write a query, the basic syntax of SPARQL queries needs to be followed. RDF is a data model of graphs built out of triples. A triple includes *subject*, *predicate* and *object*, where the predicate describes the relationship between the subject and the object. The structure of a SPARQL query comprises, in order: 1) prefix declarations for abbreviating the URI, 2) a dataset definition, stating what RDF graph is being queried, 3) a result clause, that identifies what information to return from the query, 4) a query pattern that specifies what to query and 5) query modifiers (that are optional) for rearranging the queries. Figure 8 below shows the general structure of a SPARQL query.

```
# prefix declarations
PREFIX foo:
<http://example.com/resources/>
 # dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
    ...
}
# query modifiers
```

Figure 8: Structure of a SPARQL Query

Example queries based on the specification of UML for the Transceiver API are 1) a query to return the instances that implement API's, that are in turn used by the Waveform Application object. This is done by calling all instances that are part of the Transceiver Subsystem and then filter the specific instances that have "implementAPI" as a property, and further set a constraint for specific instances that are used by the Waveform Application. 2) a query to check whether there are APIs that are needed but not implemented in the CRO. We do this by calling all instances of *Process* and then get the *Methods* that use the API. We can constrain this by making sure we only show the methods that have the property of *useAPI* and not *implement API*.

Figure 9 shows the UML class diagram of the class Transmit Channel that implements the two API's: Transmit Control and Transmit Data Push. The UML relationship that is shown here is the Transceiver Subsystem (Figure 3) that comprises the Transmit Channel and the Receive Channel. The Transmit Channel class is mapped to the CRO as a *Component* that is a subclass of *Object*. The API's are mapped to the CRO as *Methods* that are subclasses of *Process*. The syntax of the query used to find the *Components* that *implementAPI* is shown in Figure 10. The query is in the form of a triple, where *implementAPI* acts as the predicate between the subject/range C and object /domain A.

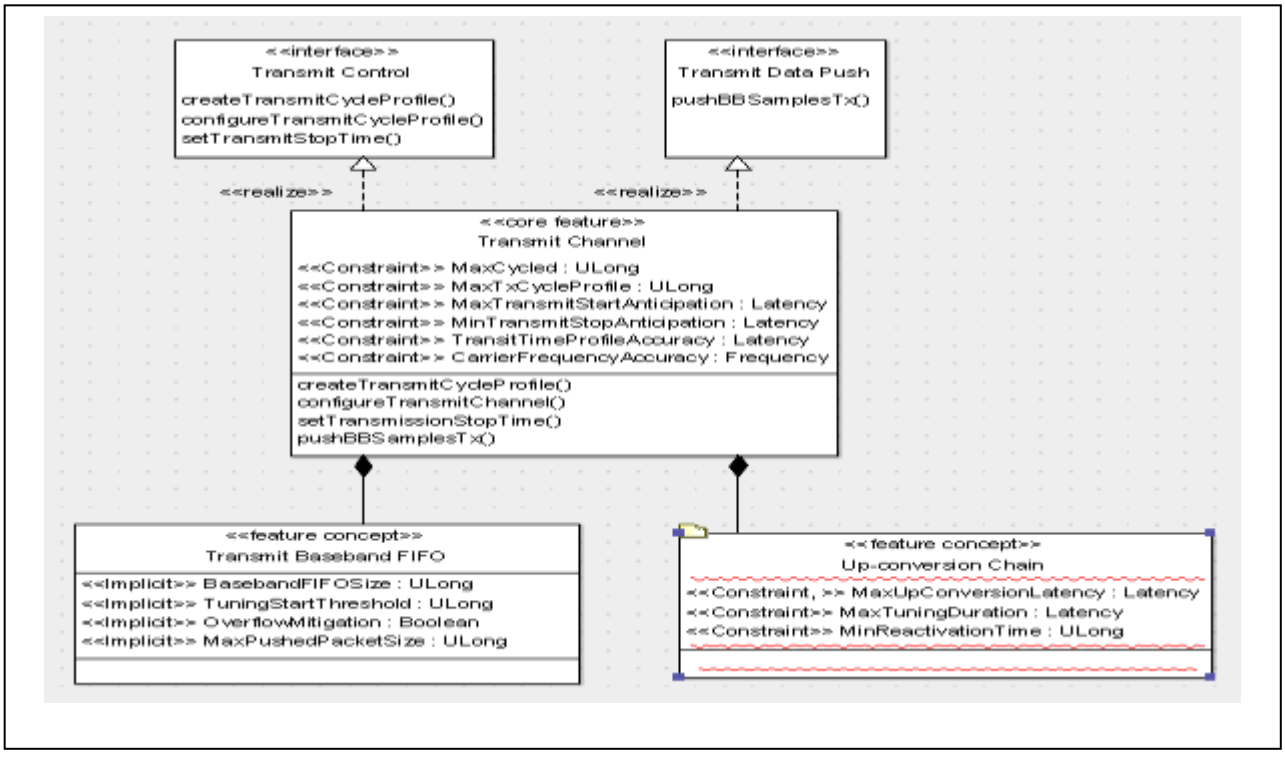

Figure 9: Transmit Channel

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

For the query in Figure 10, the inference engine first needs to instantiate the classes, and map the relationship between the Transmit Channel, and the Transmit Control and Transmit Data Push. The class Component was instantiated with C1, C2 and the class API was instantiated with X, Y and Z. Each instance of Component (C1 and C2) is then mapped to the instances of API (X and Y) using the property *implementAPI*. When we run the query shown in Figure 10, C and A act as the subject and object and the result is a table with the C and A as columns and show that C1 implements X and C2 implements Y. This is shown in Figure 11.

```
SELECT ?C ?A
WHERE { ?C :implementAPI ?A}
```

Figure 10: SPARQL Query

This query example is obviously very simple. However, large APIs may be very complex and thus many different queries can be executed and the results can be assessed by the API specifiers. In general, queries can be written to check if the dependencies have been implemented correctly from the UML to the CRO.

Results

C	A
C2	Y
C1	X

Figure 11. SPARQL Query Result

4. CONCLUSION AND FUTURE WORK

In conclusion, this paper aims to show the advantages of having an Ontology based API that is written in OWL in addition to a UML specification, since in this way the specification can be tested automatically by an OWL reasoner. Thus the aim was to show some of the advantages of complementing the power of UML with that of OWL. Although it was not shown in this paper, the API specifications can even be run without the need to implement them in a programming language. This can be achieved by using a standard OWL API for this purpose (as done in our work referenced in this paper). In the future we will continue working on the following aspects: 1) prototyping the various API's of the SCA in OWL, 2) checking for inconsistencies in not only the associations but also the dependencies that exist in UML, 3) implementation of the single ontology-based API that will allow for mapping any of the existing UML specified APIs to the CRO ontology.

5. REFERENCES.

[1] C.Magsombol, C. Jimenez, D.R. Stephens, Joint Tachtical Radio System - Application Program Interfaces. Joint Program Technical Report Executive Ofice, Joint Tachtical Radio System Standards, San Diego, CA.

[2] R.Studer, V.R Benjamin, D.Fensel, "Knowledge Engineering: Principles and Methods", Data and Knowledge Engineering pp. 161-197, 198

[3] C. Masolo, S. Borgo, A. Gangemi, "DOLCE: a Descriptive Ontology for Linguistics and Cognitive Engineering:., Institute of Cognitive Sciences and Technology, Italian National Research Council.

[4] "API Position Paper" Software Defined Radio Forum, System Interface Working Group Document Number: SDRF -03-A-0005-V0.0 July 19th, 2003

[5] "Transceiver API Specification" Software define radio Forum, Document Number: SDRF-08-S-008-V1.0.0 January 28th, 2009

[6] Object Management Group, Object Data Metamodel (ODM), version 1.0, OMG Document Number: formal/2009-05-01 Standard document URL: http://www.omg.org/spec/ODM/1.0

[7] S. Li, M. M. Kokar, D. Brady, "Developing an Ontology for the Cognitive Radio: Issues and Decisions", SDR Forum Technical Conference, Dec. 2009.

[8] E. Nicollet, L. Pucker, Standardizing Transceiver APIs for Software Defined and Cognitive Radio. RF Design Magazine, February 2008.

[9] G. Hillariet, ATL Use Case - ODM Implementation (Bridging UML and OWL). SIDo Group,February 2007.

[10] Object Management Group, "UML 2.0 Superstructure Revised Final Adopted Specification," *OMG Specification*, Oct. 2004.

[11] M.K. Smith, C. Welty, and D.L. McGuinness, eds., "OWL Web Ontology Language Guide", *World Wide Web Consortium (W3C) recommendation*, Feb. 2004, 53. http://www.w3.org/TR/2004/ REC-owl-guide-20040210/.

[12] D. Harel and B. Rumpe, "Modeling Languages: Syntax, Semantics and All That Stuff - Part I: The Basic Stuff", *tech. report MCS00-16*, Faculty of Mathematics and Computer Science, The Weizmann Inst. of Science, Israel, 2000.

[13] Specification and Description Language (SDL), International Telecommunication Union, ITU – T, Z.100 (11/89)

[14] WIF Forum MLM Working Group, "Description of Cognitive Radio Ontology v.1.0", 2010.

[15] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993.