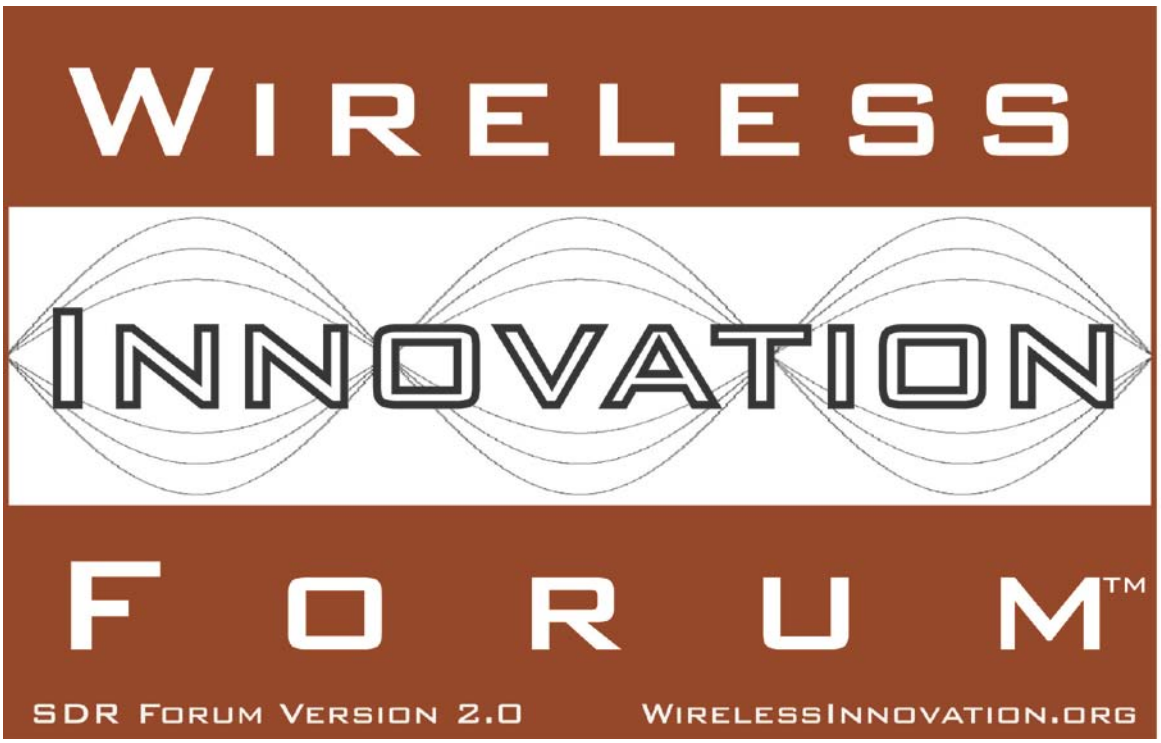

Description of the Cognitive Radio Ontology

Document WINNF-10-S-0007

Version V1.0.0

30 September 2010

TERMS, CONDITIONS & NOTICES

This document has been prepared by the Modeling Language for Mobility (MLM) Work Group to assist The Software Defined Radio Forum Inc. (or its successors or assigns, hereafter "the Forum"). It may be amended or withdrawn at a later time and it is not binding on any member of the Forum or of the MLM Work Group.

Contributors to this document that have submitted copyrighted materials (the Submission) to the Forum for use in this document retain copyright ownership of their original work, while at the same time granting the Forum a non-exclusive, irrevocable, worldwide, perpetual, royalty-free license under the Submitter's copyrights in the Submission to reproduce, distribute, publish, display, perform, and create derivative works of the Submission based on that original work for the purpose of developing this document under the Forum's own copyright.

Permission is granted to the Forum's participants to copy any portion of this document for legitimate purposes of the Forum. Copying for monetary gain or for other non-Forum related purposes is prohibited.

THIS DOCUMENT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS DOCUMENT.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

This document was developed following the Forum's policy on restricted or controlled information (Policy 009) to ensure that that the document can be shared openly with other member organizations around the world. Additional Information on this policy can be found here: http://www.wirelessinnovation.org/page/Policies_and_Procedures

Although this document contains no restricted or controlled information, the specific implementation of concepts contain herein may be controlled under the laws of the country of origin for that implementation. Readers are encouraged, therefore, to consult with a cognizant authority prior to any further development.

Wireless Innovation Forum ™ and SDR Forum ™ are trademarks of the Software Defined Radio Forum Inc.

Table of Contents

List of Figures

List of Tables

Contributors

Bruce Fette
General Dynamics C4 Systems

James Rodenkirch
Diversified Technology, Inc.

Vince Kovarik
Harris Corporation

Peter G Cook
Hypres Inc

Filip Perich
Shared Spectrum Company

Joe Mitola
Stevens Institute of Technology

Eric Nicollet
Thales Communications

John Chapin
Vanu

Mitch Kokar
VIStology, Inc.

Grit Denker
Guest Member

Shujun Li
Guest Member

Dennis Stewart
Guest Member

Many other individuals both within and outside of the Wireless Innovation Forum helped with the creation of this document.

Description of the Cognitive Radio Ontology

1 Introduction

The Cognitive Radio Ontology is an important part of the Modeling Language for Mobility (MLM) project undertaken by the MLM Working Group of Wireless Innovation Forum (WINNF).

The MLM project focuses on the signaling between advanced radio systems to support next generation features of vertical and horizontal mobility, spectrum awareness, dynamic spectrum adaption, waveform optimization, capabilities and feature exchanges.

The objectives of the MLM project include:

(1) Development of use cases for wireless communication in which the MLM language can facilitate flexible communication,
(2) Development of Cognitive Radio Ontology (CRO) that is capable of expressing structural, functional and behavioral aspects of models for wireless communication,
(3) Corresponding signaling plan, requirements and technical analysis of the information exchanges that enable these next generation features,
(4) Policies and rules for policy based radio control,
(5) Ontology extensions needed to support policy based radio control.

This document presents the Cognitive Radio Ontology. This ontology includes:

- *Core Ontology (covering basic terms of wireless communications from the PHY and MAC layers)*
- *Concepts needed to express the use cases; only the use cases that relate to the PHY and MAC layers are included*
- *Partial expression of the FM3TR waveform (structure and subcomponents, FSM)*
- *Partial expression of the Transceiver Facility APIs*

It is expected that the Cognitive Radio Ontology and the MLM Language developed in this project will provide opportunities for development of interoperable radios by independent vendors and lead to specifications/standards for data exchange to support the next generation capabilities.

2 Principles of Modeling

2.1 Top-Level Classes

An upper ontology defines the most general concepts that are the same across different domains. Choosing an appropriate upper ontology as a reference model is beneficial since this will help merging different ontologies into one so that the common classes and properties (relations) are mapped correctly.

From among the well-know upper level ontologies, we chose DOLCE, the Descriptive Ontology for Linguistic and Cognitive Engineering [1], as our reference model. DOLCE is based on the fundamental distinction between *Endurant*, *Perdurant* and *Quality*.

Endurant, also known as *Object* in our ontology, refers to the entity that is wholly presented at any given snapshot of time. Examples include material objects such as a piece of paper or an apple, and abstract objects such as an organization or a law.

Conversely, Perdurant, also known as *Process* in our ontology, is the entity that can be presented only partly at any snapshot of time. A process can have temporal parts and spatial parts. For example, the first movement of a symphony is a temporal part of a symphony, whereas the symphony performed by the left side of the orchestra is a spatial part of a symphony. In both cases, a part of a process is also a process itself.

At this point, we have identified the following relationships between Object and Process (see Figure 1).

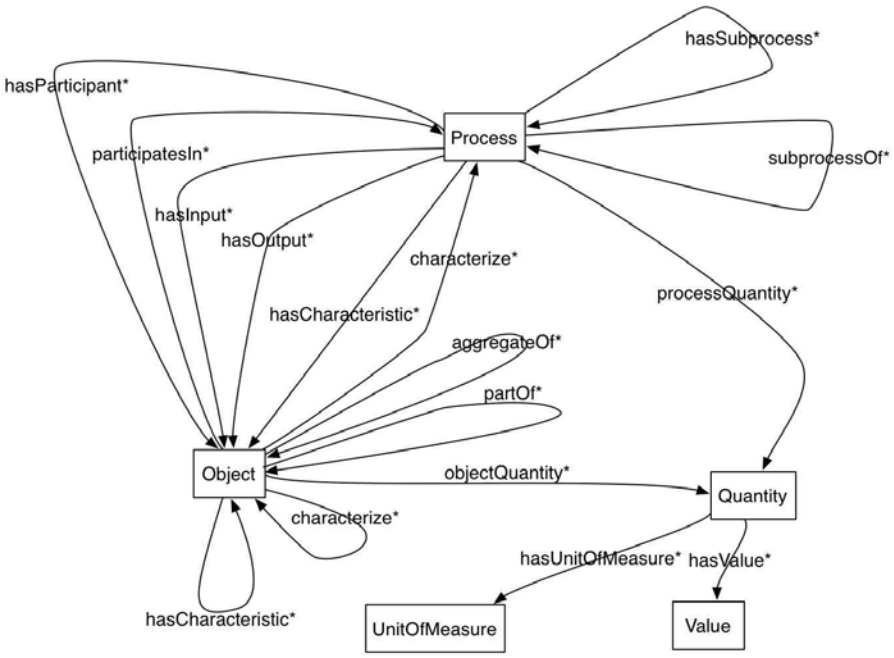

Figure 1 Top-Level Classes of CRO

(1) An object cannot be a part of a process, but rather *participate* in a process. For example, a person is not a part of running, but rather participates in running.

(2) The input and output of a process are objects. For instance, the input of modulation is a signal, where modulation is a process and signal is an object.

(3) The capabilities of an object are a collection of processes. For example, a radio has the capabilities of transmitting and receiving. Here, a radio is an object; transmitting and receiving are processes.

(4) The characteristics of an object or a process can be represented as objects. For instance, one of the characteristics of a transmitter is represented as *TxChannelTransferFunction*.

Qualities are the basic attributes or properties that can be perceived or measured. Qualities cannot exist on their own; they must be associated with either an object or a process. All the qualities have values and some qualities have unit. The qualities without units are represented as data-type properties. The qualities with units are associated with a type of quantity.

Quantity is a representation of a property of an object. In other words, quantity is a representation of quality. For instance, a physical quantity represents a property of a physical object. Quantity carries three types of information: the type of the quantity (e.g., mass, length), the magnitude of the property (typically a real or integer number) and the unit of measurement associated with the given magnitude (e.g., [*kg*], [*m*]). In this ontology, quantity is a top-level class; it is further divided (sub-classified) into different types, such as length, frequency, time, etc. Each *quality* is associated with a unit and a value.

Note that there is no explicit *Quality* class in our ontology. Instead, we use *objectQuantity* and *processQuantity* to represent the quality of an object or a process, as shown in Figure 1.

There are two perspectives to representing the quality of an object or a process depending on whether the quality has a unit or not. If the quality has a unit then the quality is represented as a sub-property of *object-type* property *objectQuantity*. If it does not have a unit then it is represented as a *data-type* property.

For example:
(1) *hasWeight* is a quality of *Student*; its unit is *kilogram*. Therefore, *hasWeight* is represented as a sub-property of object-type property *objectQuantity*. The domain of *hasWeight* is *Student*; the range is *Weight* (*Weight* is a sub-class of Quantity).
(2) However, *studentID* is a quality of *Student*; it is represented as a data-type property. The domain of *studentID* is *Student*; the range is *Integer* (*Integer* is one of the built-in data types).

The same principle can be applied to represent a quality of a process.

According to the classification described above, the top-level classes in our ontology are shown in Figure 1, including (1) Object, (2) Process, (3) Quantity, (4) Value, and (5) UnitOfMeasure.

2.2 Further Distinction: Object and Process

In this section, we are going to use some examples to further discuss the distinction between object and process. Table 1 shows an example list of objects and processes. All the examples are basic concepts within the cognitive radio domain.

Table 1 Examples of Object and Process

Object		Process	
Alphabet	ChannelEncoder	State	ChannelCoding
AlphabetTableEntry	Detector	Transition	Detection
Channel	Modulator	Event	Modulation
ChannelModel	SourceEncoder	Action	SourceCoding
Component	Transceiver	AIS	Transceiving
Port	PNCode	Protocol	PNCodeGeneration
Agent	Packet	API	Multiplexing
Goal	PacketField	Method	…
DetectionEvidence	Network		
Signal	NetworkMembership		
Sample	Role		
Symbol	……		

2.2.1 Physical Object vs. Non-Physical Object

The distinction between physical object and non-physical object depends on whether an object has spatial qualities. All the objects exist in time; but not all of them exist in space. The objects that exist in time and space, i.e. the ones with spatial location, are physical objects [2]. Typically, the term physical object and material object are interchangeable. Conversely, non-physical objects only exist in time. For instance, signal is a physical object because it can be measured through time and space whether it is the signal conducted in the radio or the signal radiated in space.

Channel is the physical transmission medium, though it may not be visible by human eyes, it does indeed exist in both time and space and thus is a physical object. *ChannelModel* is a mathematical model that represents the characteristics of the channel. Most abstract mathematical concepts such as equations, functions are non-physical objects. *Goal* is the objective that an object intends to achieve. *Role* refers to what position a network member has in the network, e.g. master, slave or peer. Both goal and role are non-physical objects. *Detector* can refer to either physical object or non-physical object depending on what detector refers to. If detector refers to a physical device, e.g. a GPS as a location detector, then detector is classified as a physical object. This physical detector is visible and tangible; it has height and mass that represent its spatial qualities. However, detector may also refer to the software module that performs the detection functionalities. In this case, detector is a non-physical object. The same

methodology can apply to the analysis of some other objects listed in Table 1. For some concepts, if a precise definition is not given, then it is difficult to say whether it is a physical object or a non-physical object. Therefore, in this version, we do not further distinguish object class as physical object and non-physical object.

2.2.2 Object vs. Process

2.2.2.1 Example1: Alphabet, Modulator, Modulation

The relationships among *Alphabet*, *Modulator* and *Modulation* are good examples to show the relationship between Object and Process.

Modulation is a process of varying one or more properties of a high frequency periodic waveform, called the carrier signal, with respect to a modulating signal. It usually takes digital signal as input and converts it to analog signal. Then the analog signal will be transmitted to the wireless channel. The changes in the carrier signal are chosen from a finite number of M alternative symbols, which is called *alphabet*.

Alphabet, also known as *modulation alphabet*, is often represented on a constellation diagram. A constellation diagram represents the possible symbols that may be selected by a given modulation scheme as points in the complex plane. The coordinates of a point on the constellation diagram are the symbol values. If the alphabet consists of $M = 2^N$ alternative symbols, then each symbol represents a message consisting of N bits. The index of each symbol value implies the bit pattern for that symbol. In real applications, *alphabet* is actually a lookup table that has the index and symbol value for each symbol. Regardless whether *alphabet* is a lookup table or a collection of symbols, *alphabet* presents itself as a whole at any snapshot of time; *alphabet* is a non-physical object.

Modulator refers to either an electronic device or a software module that performs modulation. In the former case, *modulator* is a physical object with input ports and output ports; in the latter case, *modulator* is a non-physical object that encapsulates a set of related functions, data and interfaces.

2.2.2.2 Example 2: Specification vs. Implementation

Air Interface Specification (AIS) refers to "the set of transformations and protocols applied to information that is transmitted over a channel and the corresponding set of transformations and protocols that convert received signals back to their information content" [3].

Typically, the specification of AIS is a document that establishes uniform criteria, methods, processes, etc; therefore, it is a non-physical object. In the DOLCE taxonomy, the specification of AIS can be further classified as a non-agentive social object, which is a sub-class of non-physical object. If two radios want to communicate with each other, both of them should implement the processes and methods in the AIS specification, though the details of the implementation may vary. Therefore, the implementation of AIS is a process.

Application Programming Interface (API) is a similar concept, it refers to an abstraction that a software entity provides of itself to the outside in order to enable interaction with other software entities. Typically, API contains the abstract description of a set of classes and functions. The software that provides the functions described by an API is said to be an implementation of the API. Therefore, it can be said that the specification of API is an object whereas the implementation of the API is a process.

We use AIS and API as examples to demonstrate the difference between specification and implementation because they have something in common. Both of them are interfaces that provide a "standardization" to enable interaction between two objects. This standardization is an agreement that both of the objects must satisfy.

In general, we consider the specification of such an interface as an object whereas the implementation of this interface as a process.

In our ontology, we have both API and AIS categorized as sub-classes of process. It is assumed that the term API and AIS refer to the implementation, though the naming may not reflect this assumption.

2.3 Part-Whole Relationship

2.3.1 Aggregation vs. Composition

In UML (the Unified Modeling Language), *aggregation* and *composition* are two different types of *association*; both of them represent a part-whole relationship. There is a distinction between aggregation and composition. *Aggregation* refers to the association relationship between two classes when a class is a collection or container of another class, but the contained class does not have a strong life cycle dependency on the container, i.e. when the container class is destroyed, its contents are not [15][4]. For instance, *AIS* consists of one or more *protocols* for each layer that perform the layer's functionality. When the *AIS* no longer exists, its contained *protocols* are still there. Therefore, *AIS* is an aggregation of *protocols*. Conversely, *composition* has a stronger life cycle dependency between the container class and the contained class. When the container class is destroyed, its contents are destroyed, too. For instance, an *alphabet* table has several *alphabetTableEntry*, each *alphabetTableEntry* refers to a row in the table. When the *alphabet* table is destroyed, all the rows in that table no longer exist.

Figure 2 Naming Schemes for Aggregation and Composition

In UML 2, properties (associations) are formalized in the UML meta-model using the meta-classes *Association, Property, Class* and *DataType*. *Association* is an aggregation of two or more *Property* meta classes. One of the Property classes is linked with a *Class* representing the domain of the association. Depending on whether the association range are objects or data types, the other *Property* classes are linked with either *Class* or *DataType* meta classes. UML uses the *isComposite* Boolean-type meta-property of the *Property* meta-class to specify that a given aggregation is composite (strong aggregation). Since it is not possible to represent the property of a property in OWL, we use different naming schemes to distinguish between *aggregation* and *composition.* All the aggregation properties start with "aggregateOf" followed by the name of the range class. All the composition properties start with "compositeOf" followed by the name of the range class. Figure 2 shows an example to illustrate this.

2.3.2 Ordering of the Contained Entities

An instance of a class may contain an ordered collection of instances of other classes. The order of the contained instances must be explicitly represented. For instance, a *packet* is a composite of a sequence of *packetField.* The ordering of the *packetField* is defined in the protocol. In this ontology, we use property *append* to represent the ordering of the contained instances, i.e. an instance can be appended to another instance. For instance, in the packet frame structure shown in Figure 3, *preamble, destination address, source address* and *control field* are instances of *packetField* class. Each of the *packetField* is appended to another *packetField*.

Preamble	Destination Address	Source Address	Control Field	…..

Figure 3 Packet Frame Structure

2.3.3 Examples of Part-Whole Relationship

The following examples will further illustrate how to represent part-whole relationships in this ontology.

2.3.3.1 Example 1: Alphabet and AlphabetTableEntry

In this example, *Alphabet* is a composite of *AlphabetTableEntry*. Both of them are sub-classes of *Object*. Table1 is an instance of the class Alphabet. Row1 and Row2 are instances of the class AlphabetTableEntry. The "instance of" relation is shown with a dashed line.

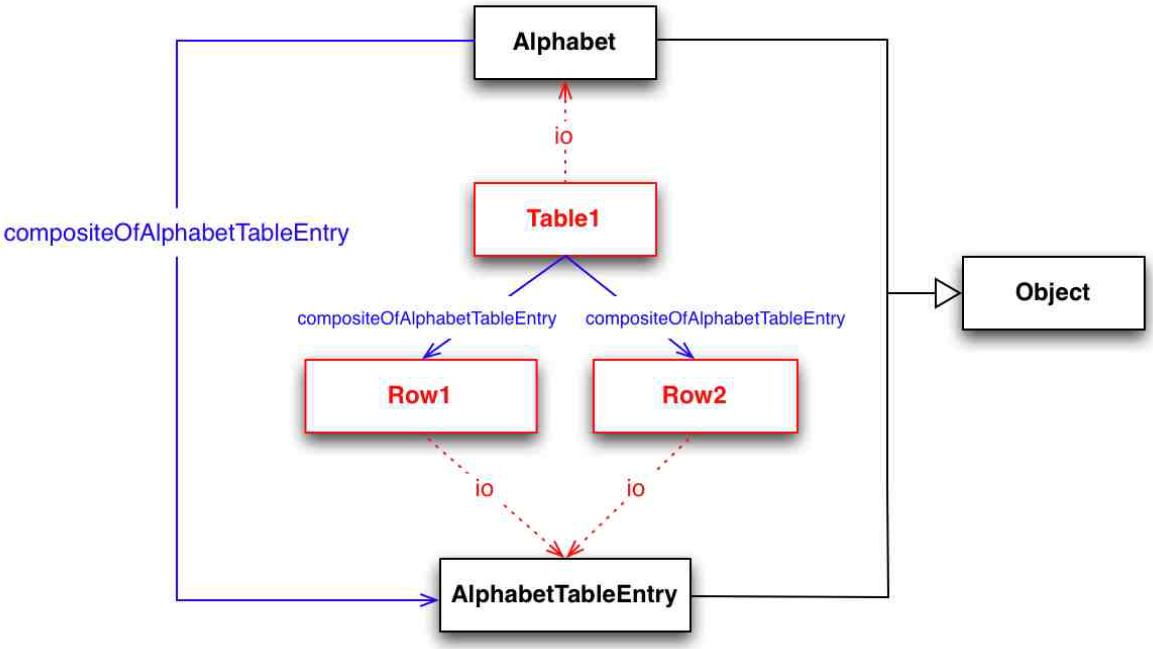

Figure 4 Example of Part-Whole Relationship (1): Alphabet and AlphabetTableEntry

2.3.3.2 Example 2: API and Method

In this example, instead of using "aggregateOf" or "compositeOf" that are used to represent the part-whole relationship between objects, we use *hasSubProcess* to represent the part-whole relationship between processes. In general, a process can have other processes as its sub-processes; in other words, a process can be a sub-process of another process. For instance, an API contains the abstract definitions of a set of methods. Both *API* and *Method* are considered as *process*. An *API* has several *methods* as its sub-processes.

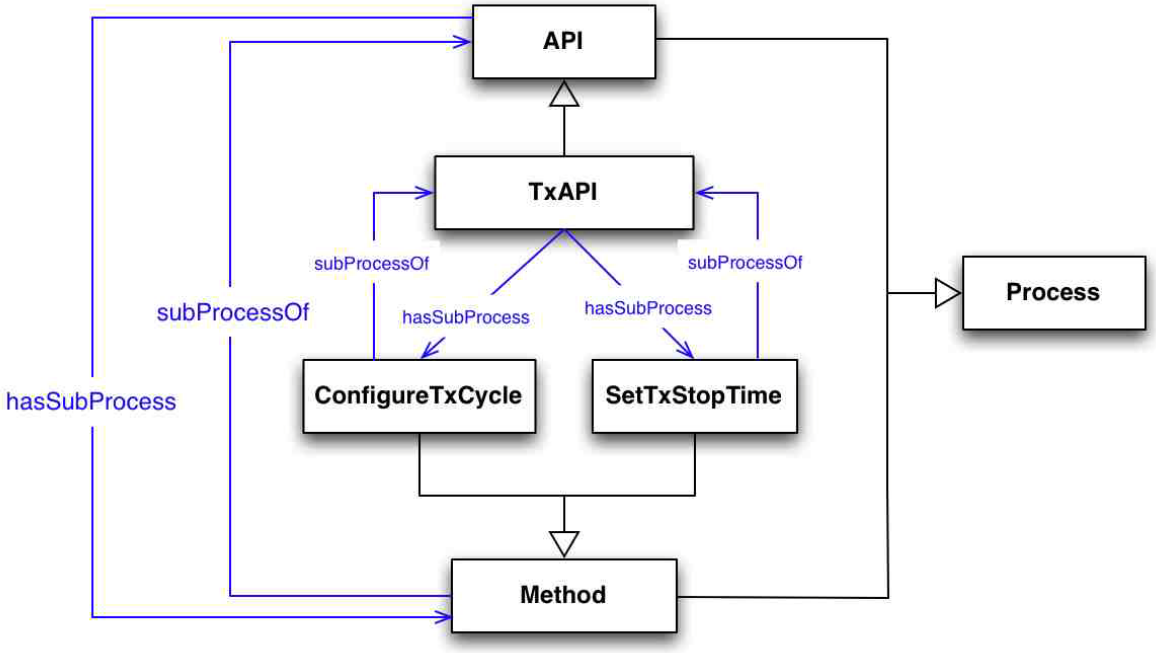

Figure 5 Example of Part-Whole Relationship (2): API and Method

2.3.3.3 Example 3: Radio and RadioComponent

This example is used to show the part-whole relationship between objects. A radio component consists of several sub-components, such as antenna and modulator. A pair of symmetric properties, *hasSubComponent* and *subComponentOf*, are used to represent the relationship between them. These two properties are sub-properties of the more general property *aggregateOf*. Although we could use aggregateOf to capture the part-whole relationship between components, we would then lose the more specific information about this relationship and, consequently, we would not be able to infer that say A and B are components from the information that they are related via the *subComponentOf* property.

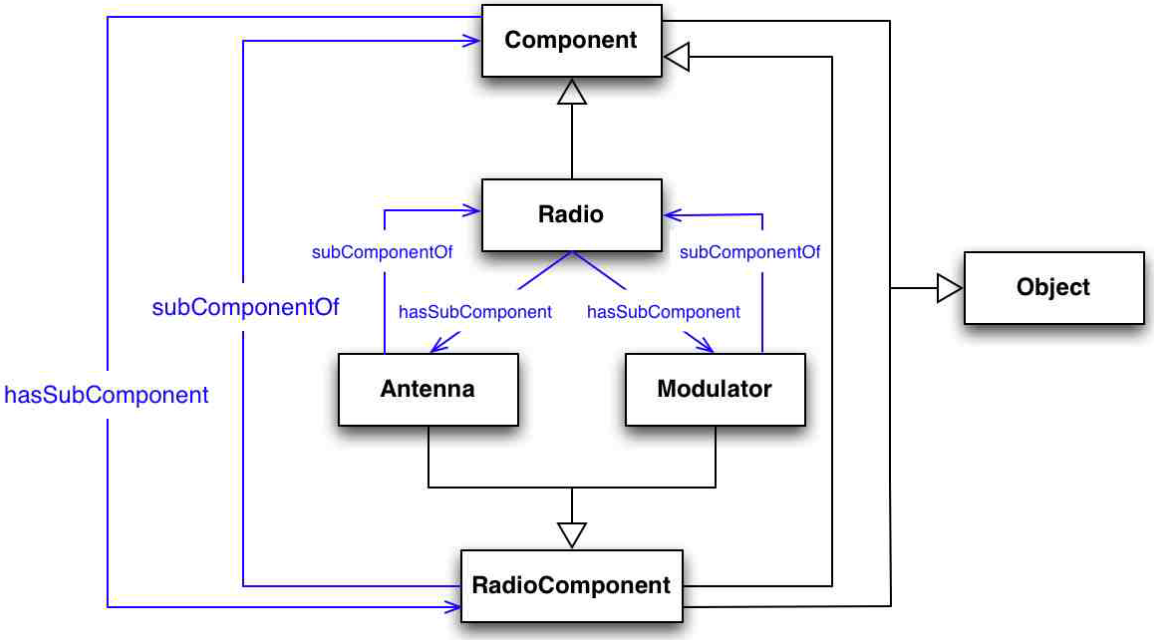

Figure 6 Example of Part-Whole Relationship (3): Radio and RadioComponent

2.3.3.4 Example 4: Signal

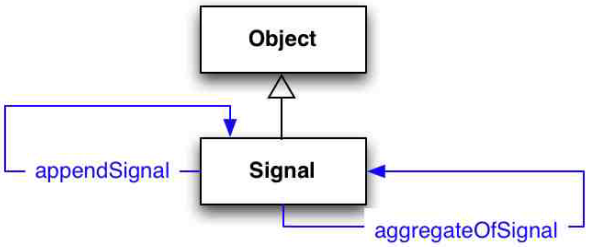

Figure 7 Example of Part-Whole Relationship (4): Signal

In some cases, the aggregateOf relationship may need to be specialized to capture some specific aspects. For instance, in some cases an aggregate of two things may lead to a different class membership of the aggregate even though the particular components come from the same class. For instance, in Example 1, *AlphabetTableEntry* is part of *Alphabet*, but it is NOT an *Alphabet*. In Example 2, *Method* is part of *API*, but it is NOT an *API*. And in Example 3, *RadioComponent* is part of *Radio*, but it is NOT a *Radio*, however, both *RadioComponent* and *Radio* are sub-classes of *Component*.

On the other hand, as shown in Figure 7, an aggregate of two signals is also a *Signal*. However, the aggregation of signals must satisfy their temporal ordering. For this reason, the *aggregateOf*

relation is specialized by adding another property *appendSignal*. The append keyword is used in this ontology to indicate the ordering of other objects.

2.4 Attributes, Properties, Parameters and Arguments

2.4.1 Attribute vs. Property

There is a need for distinguishing between property and attribute (c.f. the discussion in [5]). An attribute is a feature of an object that is independent of the context that the object is in. For instance, the size of a cup is this cup's attribute. Conversely, the property of an object depends on the context, for example, whether the cup is full or empty depends on the context, thus it should be modeled as a property.

The ontology presented in this specification is formalized in OWL (Web Ontology Language) using the Protégé tool [6]. OWL, however, does not provide any simple means for an explicit distinction between attribute and property in the sense explained above. Take *packet field* as an example. A packet consists of a sequence of packet fields. The size of a packet field is an attribute of packet field, but whether a packet field is optional or mandatory is a property since it depends on context in which a specific packet field is used. OWL only provides two types of properties: (1) object-type property, which links an individual to another individual, and (2) data-type property, which links an individual to an XML Schema data-type value (e.g. Integer, Boolean, etc.). If we only use the features provided by OWL, both *packetFieldSize* and *isOptional* should be modeled as data-type properties, i.e. *packetFieldSize* is linked to an integer value whereas *isOptional* is linked to a Boolean value [7]. For this reason, we don't distinguish between attribute and property in this ontology. All the qualities (attributes or properties) are represented as either an object-type property or a data-type property. The terms *attribute* and *property* are thus interchangeable.

2.4.2 Parameters and Arguments

The concept of parameter and argument are closely related.

In mathematics, an *argument* is an independent variable and a *parameter* is a function coefficient. For instance, in equation $aX+bY=c$, variables X and Y are *arguments* whereas the function coefficients a, b and c are *parameters*.

In computer science, *parameter* and *argument* are interchangeable.

In engineering, *attributes* of a system are the same as the properties of a system; *parameter* usually refers to combination of properties that is sufficient to describe a system response.

In our ontology, both *parameter* and *argument* are represented as either data-type or object-type properties of the *Method* class (or *Process*, or *Component*).

2.4.3 Example: Attributes and Properties of Transceiver Subsystem

Figure 8 shows an example of how to represent the properties and attributes of the Transceiver subsystem.

In this figure, *transmitCycle*, *transmitStartTime*, and *carrierFrequency* are the properties of *Transceiver*; *implementAPI* is an object-type property that shows the relationship between *component* and *API*.

transmitCycle is the integer identifier that shall be set up during the creation to a specific value, which is then incremented by one for each newly created *transmitCycle*. Since it is an integer number without any unit, it is modeled as a data-type property. The domain of this property is *Transceiver*; the range *Integer*, which is a built-in data type.

transmitStartTime refers to the transmit start time of the corresponding transmit cycle. Since it has the unit of second, it is represented as an object-type property. The domain is *Transceiver*; the range is *Time*, which is a sub-class of *Quantity*.

The way of modeling *carrierFrequency* is similar as *transmitStartTime*. It is also represented as an object-type property. The domain is *Transceiver*; the range is *Frequency*.

In general, all the properties or attributes without unit of measure will be modeled as data-type properties, the range being one of the built-in data types, e.g. *Integer*, *Boolean*, *String*. Any properties or attributes with a unit of measure will be modeled as object-type properties, the range will be one of the sub-classes of *Quantity*, e.g. *Time*, *Frequency*, *Location*. Each sub-class of *Quantity* has a *Value* and a *UnitOfMeasure*. In this way, we can specify the values for those properties or attributes that have units.

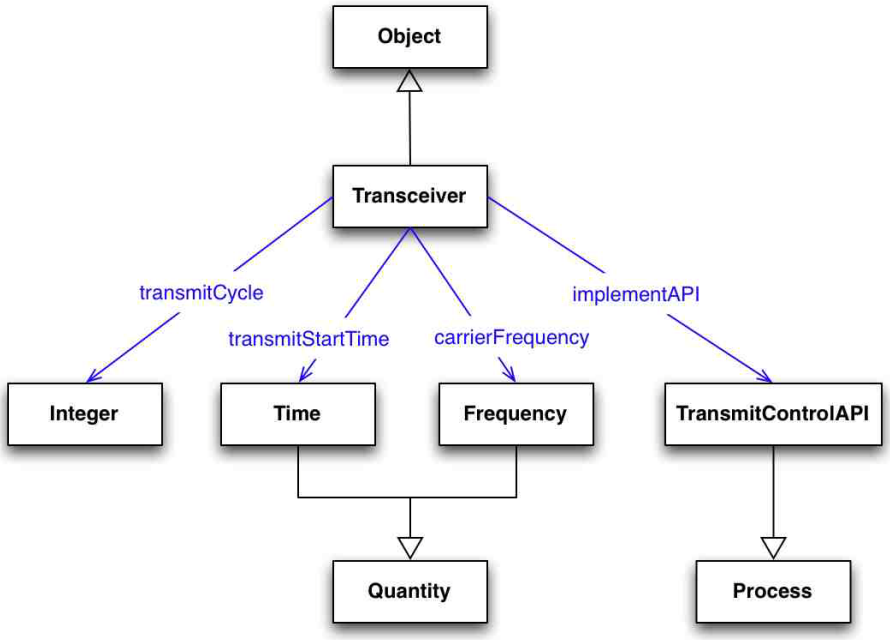

Figure 8 Representations of Properties and Attributes: Example of Transceiver

3 Object

3.1 Alphabet and AlphabetTableEntry

Alphabet is a lookup table that participates in the *Modulation* process. In digital modulation, an analog carrier signal is modulated by a digital bit stream. The changes in the carrier signal are chosen from a finite number of *M* alternative symbols, which is called *Modulation Alphabet* or *Alphabet*. Each row of the *Alphabet* table is an instance of *AlphabetTableEntry* class. Hence, the *Alphabet* is an aggregation of *AlphabetTableEntry*. The *AlphabetTableEntry* class has two properties: 1) *hasIndex*, and 2) *hasSymbolValue*, as follows:

Table 2 Example of Alphabet Table

Index (Bit Pattern)	Symbol Value
0 (00)	1+j
1 (01)	-1+j
2 (11)	-1-j
3 (10)	1-j

A modulation alphabet is often represented on a constellation diagram. A constellation diagram represents the possible symbols that may be selected by a given modulation scheme as points in the complex plane. The Real and imaginary axes are often called the in-phase (I-axis), and the quadrature (Q-axis). The coordinates of a point on the constellation diagram are the *symbol values.*

If an alphabet consists of $M=2^N$ alternative symbols, then each symbol represents a message consisting of *N* bits.

3.2 Channel

Channel refers to the physical transmission medium between the transmitter and the receiver. For example, in a cellular system, the transmission link between the Mobile Station (MS) and the Base Station (BS) can be divided into (1) Forward Channel (from BS to MS) and (2) Reverse Channel (from MS to BS). Typically, both Forward and Reverse Channel have sub-channels for the transmission of either data messages or control messages. For instance, in the IS-95 system, there are three types of overhead channels in the forward link: pilot channel, synchronization channel and paging channel. The modulation, bandwidth, data rate and the multiplexing scheme of each channel are specified in the Air Interface Specification (AIS).

The structure of *Channel* class is shown in Figure 10. The sub-classes of *Channel* are shown in Figure 9.

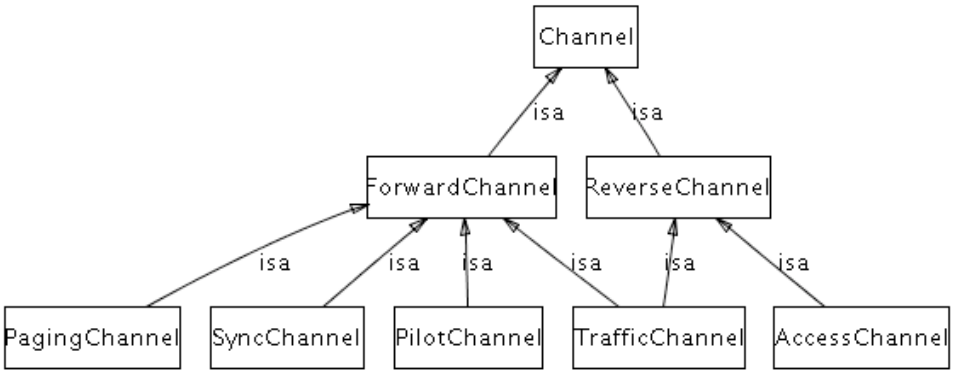

Figure 9 Sub-classes of Channel

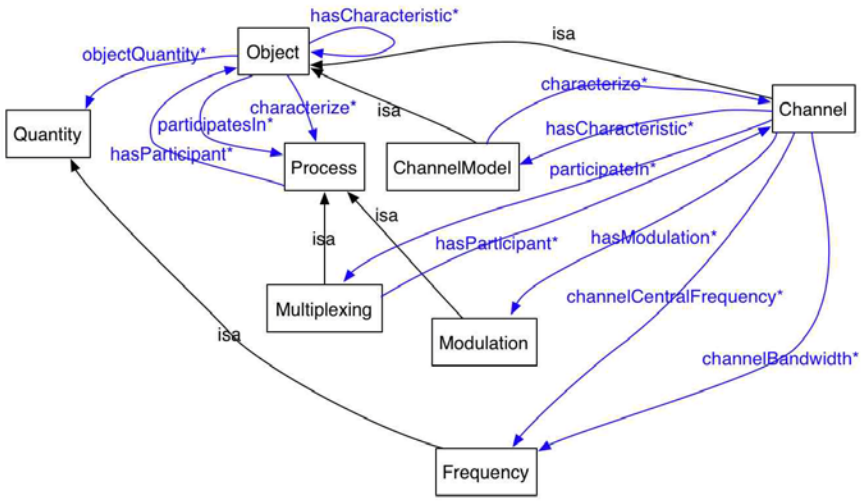

Figure 10 Structure of Channel Class

3.3 ChannelModel

Each channel is associated with one or more than one channel models. *ChannelModel* is used to represent the estimated effects of the propagation environment on a radio signal. Well known channel models include Additive White Gaussian Noise (AGN) channel, Rayleigh fading channel and Rice fading channel. The sub-classes of *ChannelModel* are shown in Figure 11.

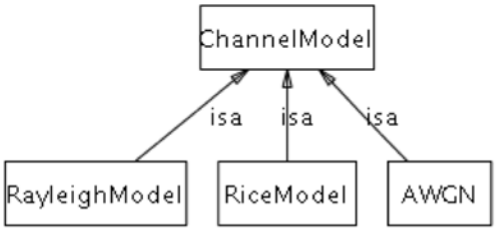

Figure 11 Sub-classes of Channel Model

3.4 Packet and Packet Field

A *Packet* is a formatted unit of data transmitted between radios.

First, a packet consists of a sequence of packet fields, thus it is an aggregation of *PacketField*.

Second, a packet field can be appended to another packet field, forming an ordered collection of packet fields.

Third, the ordering of a *PacketField*, the size of each field, and the values allowed in each field are defined in a protocol. Therefore, both *PacketField* and *Packet* are associated with a protocol. The relationships among *Packet*, *PacketField* and *Protocol* are shown in Figure 12.

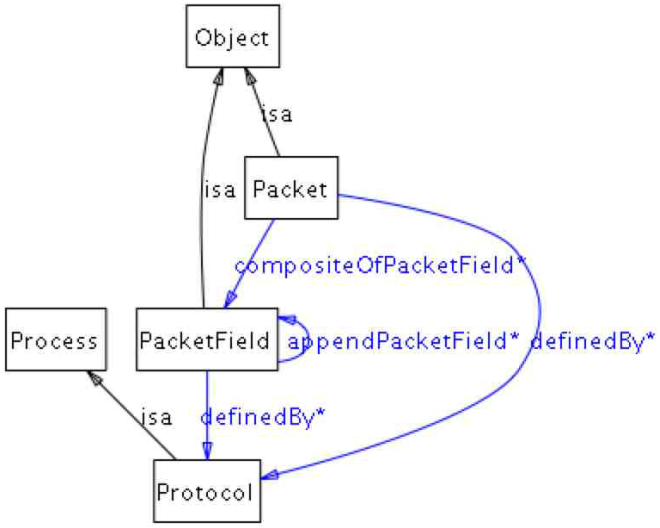

Figure 12 Relationships among Packet, PacketField and Protocol

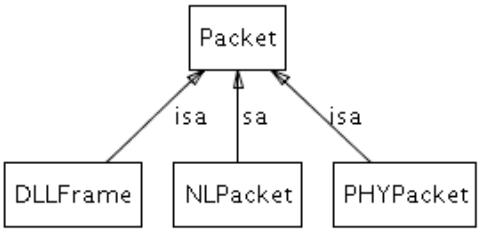

Figure 13 Sub-classes of Packet Class

In the OSI or TCP/IP model, each layer has various protocols, each of which defines the syntax and the semantics of packets. Packets of different layers may have different names. For instance, a data link layer packet is usually called a *frame*. For this reason, *Packet* is divided into several sub-classes for the lower three layers, shown in Figure 13, including *DLLFrame*, *NLPacket*, and *PHYPacket*.

There are two ways to sub-classify the *PacketField* class. First, based on the protocol that defines the packet field, the *PacketField* class can be divided into physical layer, data link layer, and network layer packet field. Second, a packet usually has a header, a payload and a trailer. Hence, the *PacketField* class can be also divided into header packet field, trailer packet field, and payload packet field. In this ontology, we represent both of the above classifications. The taxonomy of *PacketField* class is shown in Figure 14.

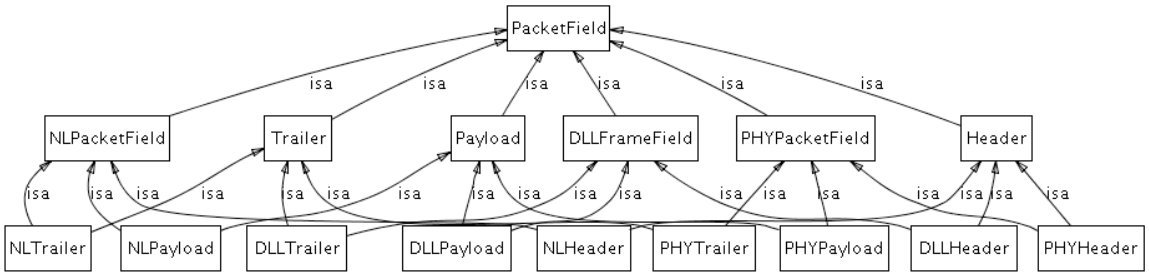

Figure 14 Sub-Classes of Packet Field

A *PacketField* class contains either user data (payload) or control information (header or trailer). The attributes of *PacketField* include *startIndex* and *fieldSize*, shown in Table 3.

Table 3 Properties of PacketField

Property	Domain	Range
startIndex	PacketField	Integer*
fieldSize	PacketField	Information

3.5 Signal

Signal is any time-varying or spatial varying quantity. There are different views on the classification of *Signal*. *Signal* can be divided into continuous and discrete signal; then further divided into quantized signal and unquantized signal. However, since our ontology is developed for the cognitive radio domain, almost all the signal processing (before the DAC or the amplifier) is implemented in software. Figure 15 shows an example of signal processing steps in a cognitive radio.

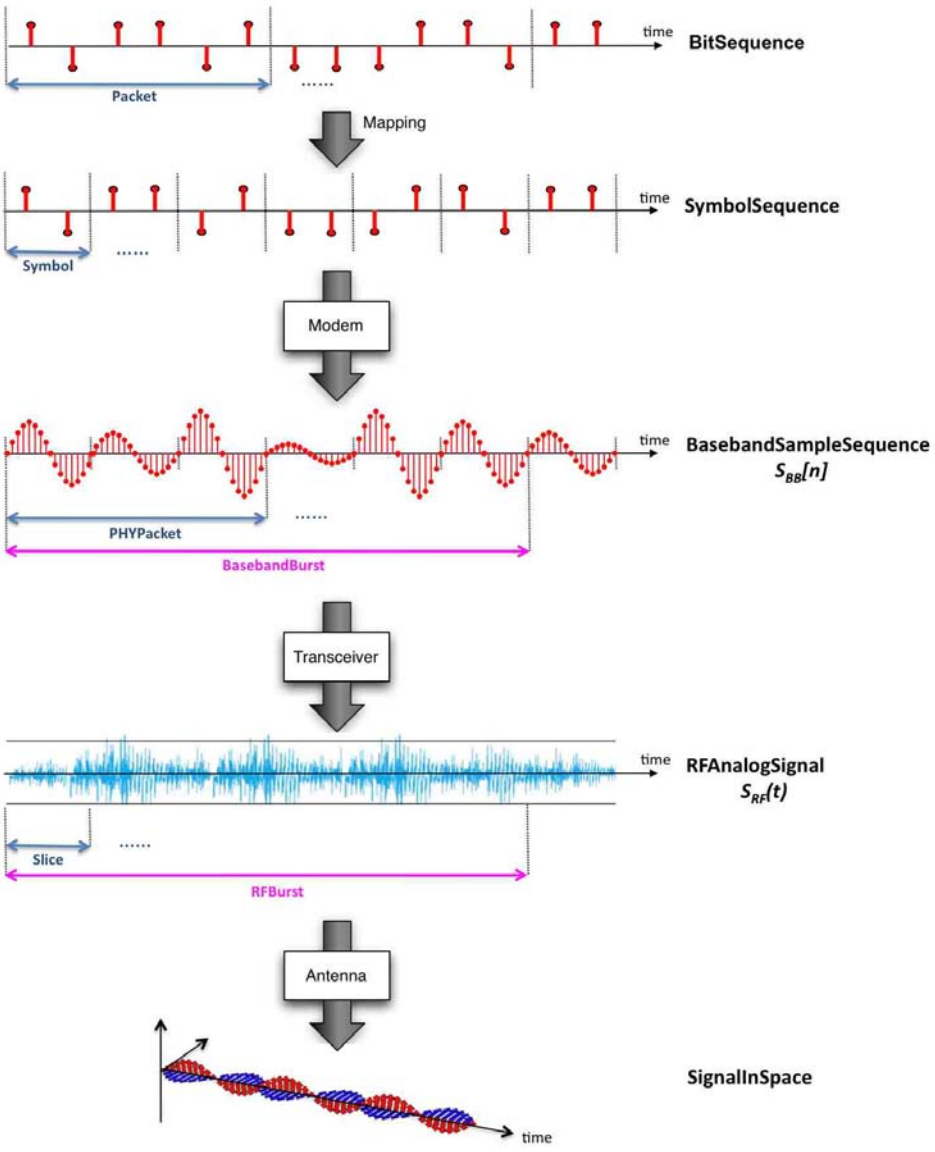

Figure 15 Signal Processing in Software Defined Radio (SDR)

(1) A *BitSequence* instance is generated, for instance, by a channel encoder. (2) The *BitSequence* is grouped into codewords, one for each symbol to be transmitted. The sequence of codewords is

represented as *SymbolSequence*. (3) *SymbolSequence* is mapped to the amplitudes of the *I* and *Q* signals, and then multiplied by the baseband frequency to produce the *BasebandSampleSequence*. (4) The *BasebandSampleSequence* is then processed in the transceiver subsystem, producing the *RFAnalogSignal*. (5) Finally, the *RFAnalogSignal* is transmitted to the air by the antenna, becoming the *SignalInSpace*.

In our ontology, we divide the *Signal* class into (1) *SignalInRadio*, (2) *SignalInSpace*.

SignalInRadio and *SignalInSpace* are disjoint classes; *SignalBurst* is NOT disjoint with *SignalInRadio* and *SignalInSpace*. Besides, *SignalBurst* is an aggregation of *Packet*.

SignalInRadio can be further divided into (1) *BitSequence*, (2) *SymbolSequence*, (3) *BandbandSampleSequence*, and (4) *RFAnalogSignal*.

Note that *Chip* is a special type of *Symbol* and is a subclass of *SymbolSequence*.

Signal class has the following basic properties: (1) Part of a *Signal* is also a *Signal*. (2) A *Signal* can be appended to another *Signal*, producing a new *Signal*. (3) *signalDuration*, and (4) *signalRate.*

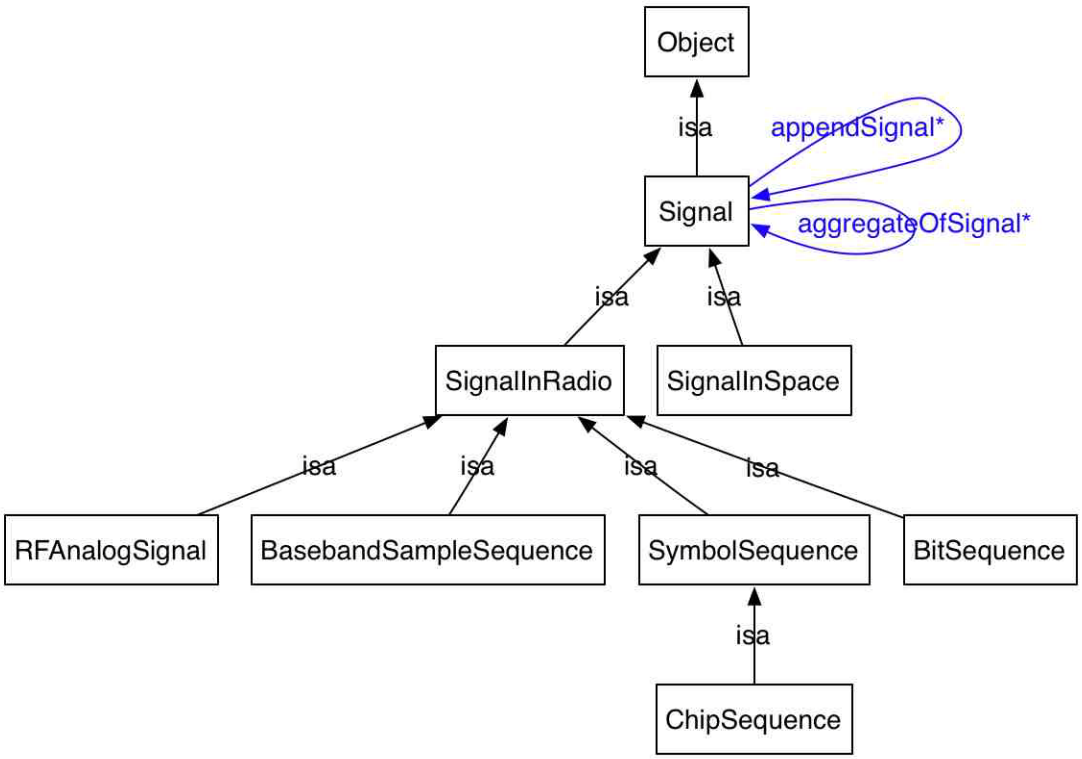

Figure 16 Classification of Signal

The classification of *Signal* class is shown in Figure 16.

Table 4 Properties of Signal

	Properties		**Domain**	**Range**
Properties of *Signal*	signalDuration		Signal	Time
	signalRate	GrossBitRate	Signal	SignalRate
		NetBitRate		
		Throughput		
		Goodput		
		SymbolRate		
		SampleRate		
		ChipRate		
		…		
Properties of *SignalInSpace*	signalPower		SignalInSpace	Power
	signalPowerDensity		SignalInSpace	PowerDensity
	signalStrength		SignalInSpace	ElectricFieldStrength

The properties of Signal and its sub-classes are shown in Table 4.

3.6 Burst

Burst is a segment of *Signal*. For radio transmission, the transmit channel of the transceiver subsystem up-converts bursts of *BasebandSampleSequence* to bursts of *RFAnalogSignal.* The transmit channel consumes the coming signal burst; stores the result in a buffer; then performs the up-conversion in real time.

A burst of *BasebandSampleSequence* is called *BasebandBurst*, consisting of several *Packets*. The length of a BasebandBurst must be a multiple of the length of a physical layer packet.

A burst of *RFAnalogSignal* is called *RFBurst*, consisting of several *Slices*. Since a *Slice* corresponds to a *Symbol*, the length of a *Slice* equals to the length of the corresponding *Symbol*. The length of an *RFBurst* must be a multiple of the length of a *Slice*.

The illustrations of *BandbandBurst* and *RFBurst* are shown in Figure 16 and Figure 17.

The relationships among Burst, Signal and Packet are shown in Figure 18.

The properties of *Burst* are shown in Table 5, including: (1) burstStartTime, (2) burstStopTime, and (3) burstLength.

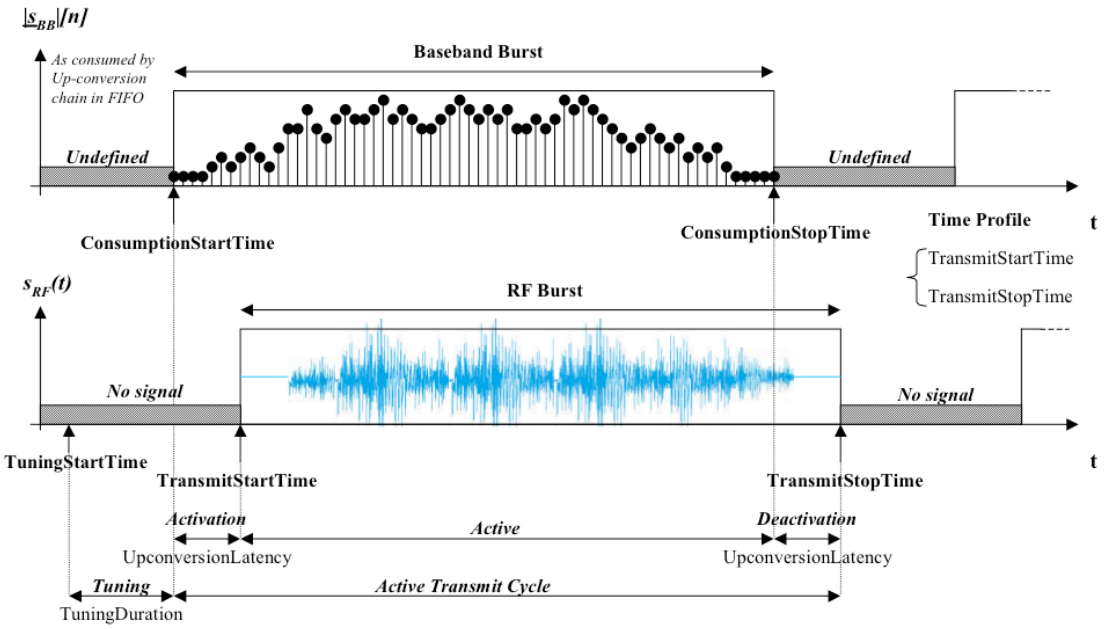

Figure 17 Illustrations of BandbandBurst and RFBurst

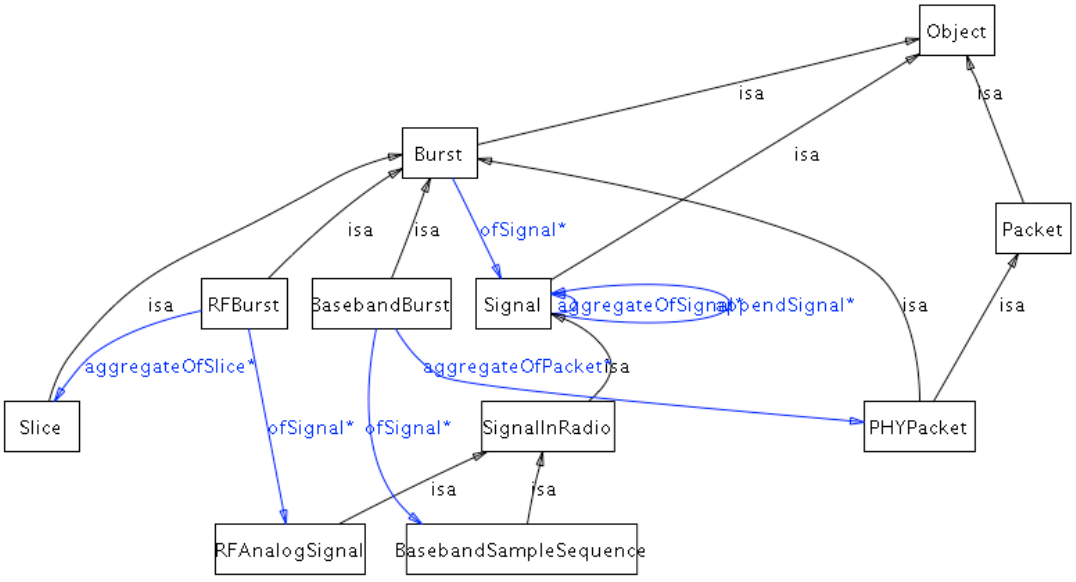

Figure 18 Relationships among Burst, Signal and Packet

Table 5 Properties of Burst

Property	Domain	Range
burstStartTime	Burst	Time
burstStopTime	Burst	Time
burstLength	Burst	Time
ofSignal	Burst	Signal

3.7 Sample

A *Sample* refers to a value taken at a point in time or space. A *Signal* is an aggregation of *Samples*. The properties of *Sample* include: (1) sample value, and (2) the time at which the sample is taken, shown in Figure 19.

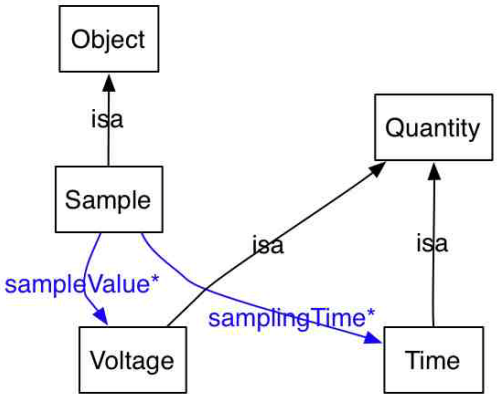

Figure 19 Properties of Sample

3.8 Symbol

The term *Symbol* is ambiguous in the sense that it is used to mean different things. (1) *Symbol* may refer to the physically transmitted signal that is placed on the channel. It is a state of the communication channel that persists for a fixed period of time [8]. For example, in passband transmission a *Symbol* usually refers to a sine wave tone, whereas in baseband transmission a symbol usually refers to a pulse rather than a sine wave tone. (2) *Symbol* may be used at a higher level and refers to one information bit or a block of information bits that will be modulated using a conventional modulation scheme such as QAM [8].

In this ontology, *Symbol* refers to the first definition mentioned above. [1]

[1] Note that the *SymbolSequence* described in the preceding paragraph refers to this definition.

Symbol properties include: (1) *InformationBitsPerSymbol*, and (2) *SymbolValue*, shown in Figure 20.

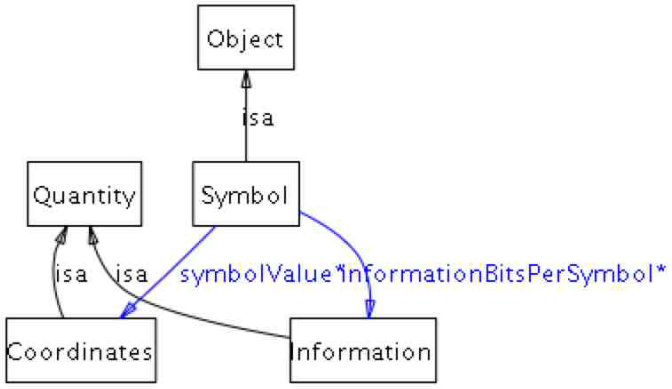

Figure 20 Properties of Symbol

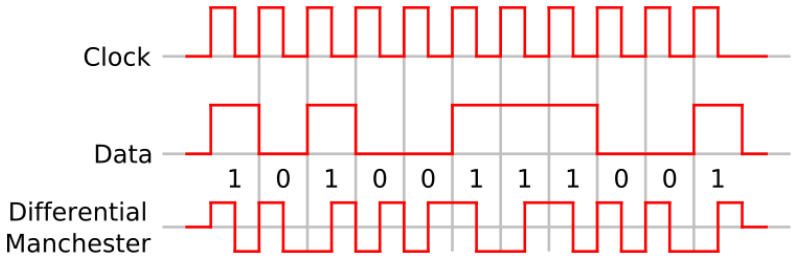

Figure 21 Illustration of InformationBitsPerSymbol

informationBitsPerSymbol refers to the number of information bits that a symbol conveys. For example, in a differential Manchester line coding shown in Figure 21, each information bit is represented by two symbol pulses, therefore, in this case, the value of *InformationBitsPerSymbol* equals to ½.

symbolValue refers to the co-ordinates of a symbol on the constellation diagram. Examples are described in Section 3.1. Note that the *symbolValue* can be a complex number or a real number, depending on which modulation scheme is used. For example, in QAM modulation, the *symbolValue* is a complex number.

In summary, the relationships among Signal, Symbol and Sample are shown in Figure 22.

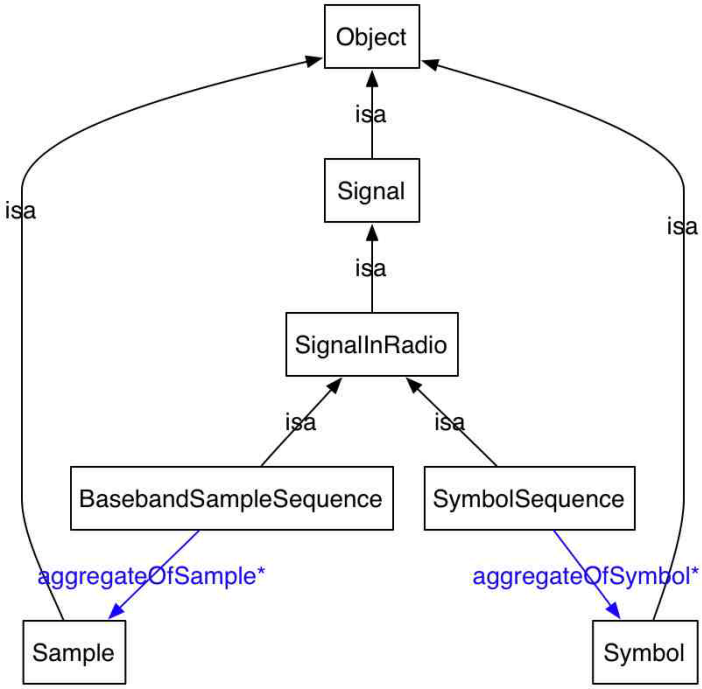

Figure 22 Relationships among Signal, Sample and Symbol

3.9 PNCode

PNCode refers to the pseudo noise code that has a spectrum similar to a random sequence of bits but is deterministically generated. *PNCode* is usually used in a direct-sequence spread spectrum system. Examples of *PNcode* include *MaximalLengthSequences*, *GoldCode*, *KasamiCode*, *BarkerCode* and *WalshCode*, shown in Figure 23.

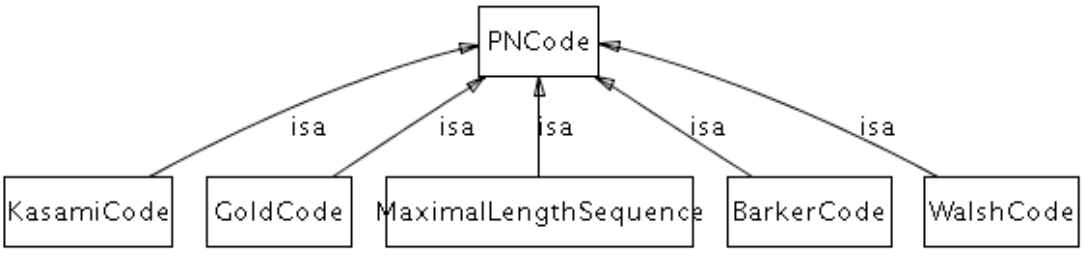

Figure 23 Sub-classes of PNCode Class

3.10 Component

3.10.1 Classification of Component

A *Component* is a self-contained part of a larger entity. It often refers to a manufactured object or a software module.

A *Component* can be part of a larger component and it can have another component(s) as its subcomponent(s). Therefore, a component has two properties: (1) *isSubComponentOf*, and (2) *hasSubComponent*.

Some components can be decomposed into smaller subcomponents whereas some components can not be decomposed. In our ontology, the components that cannot be decomposed are *BasicComponent*, e.g. Multiplier and Adder.

Radio is a complex component that consists of other *RadioComponents*.

In the current version, *Component* has three sub-classes: (1) *BasicComponent*, (2) *Radio*, and (3) *RadioComponent*.

Note that *RadioComponent* can NOT have *Radio* as its sub-component, this is represented as a restriction in our ontology. In addition, *FM3TRRadio* is a special type of radio that can be used as a test case of our ontology; we include *FM3TRRadio* as a sub-class of *Radio*.

The hierarchy of subclasses of the *Component* class are shown in Table 6.

Table 6 Sub-classes of the Component class

Component	BasicComponent	Multiplier		
		Adder		
	Radio	FM3TRRadio		
	RadioComponent	ChannelDecoder		
		ChannelEncoder		
		ChannelEstimator		
		Demodulator		
		Detector	LocationDetector	
			SignalDetector	ContinuousSignalDetector
				PulseSignalDetector
			TimeDetector	
		Equalizer		
		Fm3trComponent	Fm3tr_Dlc	
			Fm3tr_Hci	
			Fm3tr_Nwk	
			Fm3tr_Phl	
		Modulator		
		PowerAmplifier		
		Receiver		
		SourceDecoder		
		SourceEncoder		
		Transceiver		
		Transmitter		
		WaveformApplication		

3.10.2 Structure of Component

The structure of a *Component* describes the structure of the function between input variables and output variables. The function is described as a set of blocks that are connected by ports. Figure 24 shows an example of the physical layer structure component of an FM3TR radio[9].

Each *Component* has input and output ports. One component is connected to another component by ports. *Port* is an *Object*. A *Port* can be connected to another *Port* if the two ports are carrying the same type of signal. For example, if a modulator takes a digital signal as the input and outputs an analog signal, then the output port of this modulator can be connected to another port that also carries an analog signal.

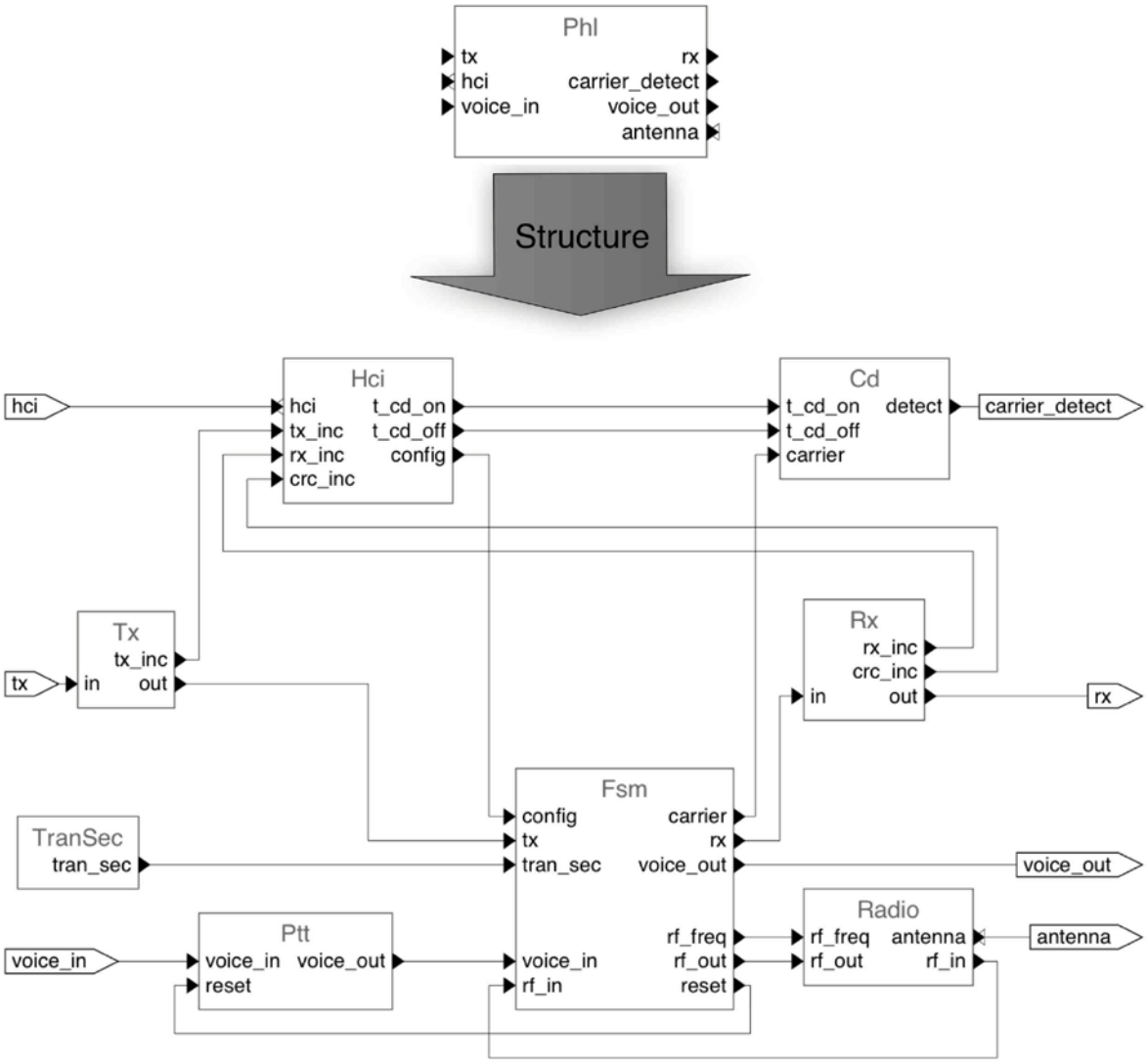

Figure 24 Example of Component Structure: Physical Layer Structure of FM3TR Radio (Source: [9])

Figure 25 shows an example of how to represent the structure of a component. Suppose component *C* has three sub-components (*C1*, *C2*, and *C3*), one input port (*P1*) and two output ports (*P9*, *P10*). First, the relationships between a component and its ports are modeled using the object-type property *hasPort*. The *hasPort* property has two sub-properties: *hasInputPort* and *hasOutputPort.* For instance, *<C1 hasInputPort p2>*, *<C1 hasOutputPort P3>* ,*<C1 hasOutputPort P4>*. Second, the relationships amnong *Ports* are modeled using the object-type property *isConnectedTo.* For instance, *<P3 isConnectedTo P5>*. Note that ports can be connected only if their port types are the same. However, we have not represented this restriction in OWL.

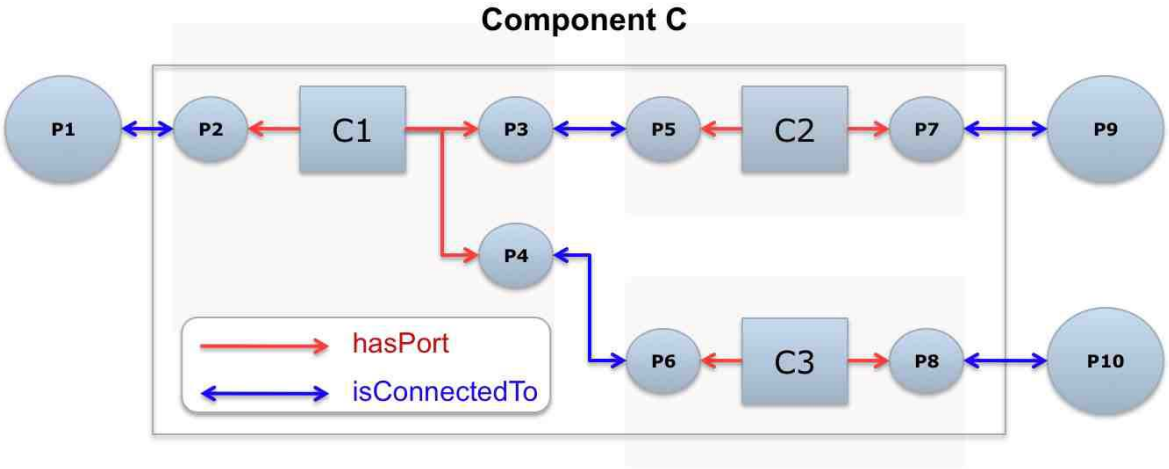

Figure 25 Representation of Component Structure

The relationships between Component and Port are shown in Figure 26.

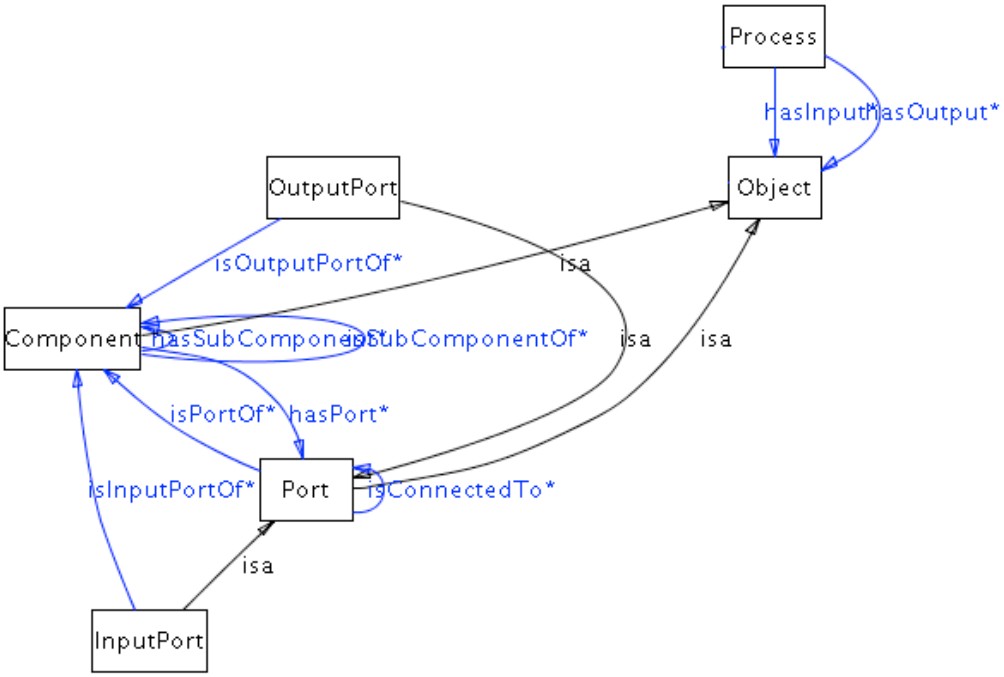

Figure 26 Relationships between Component and Port

3.10.2.1 Example: OWL Representation of FM3TR Structure

Figure 27 shows the top-level structure of an FM3TR radio. Figure 28 shows the OWL representation of this structure.

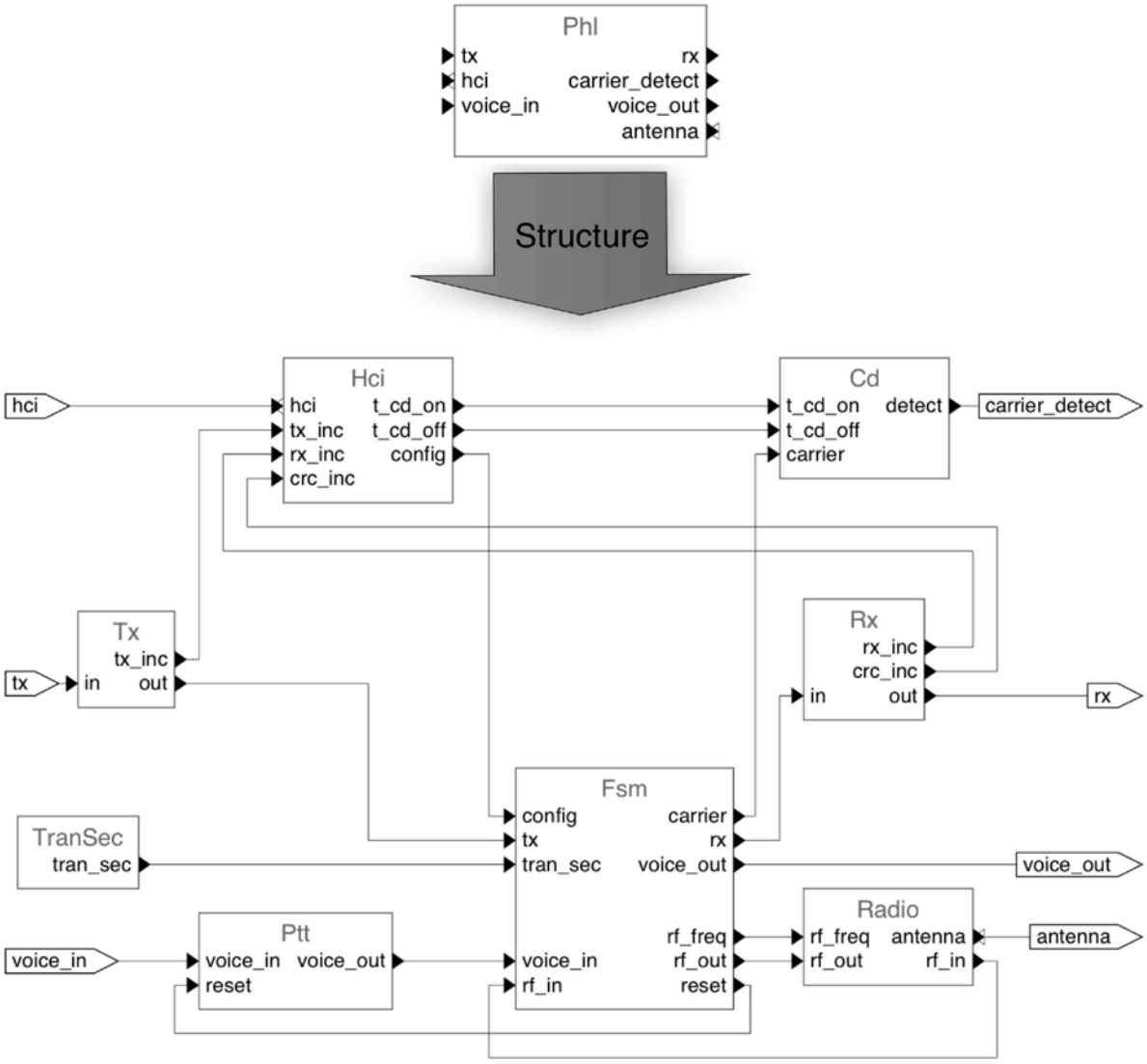

Figure 27 Top-Level Component Structure of FM3TR Radio (Source: [9])

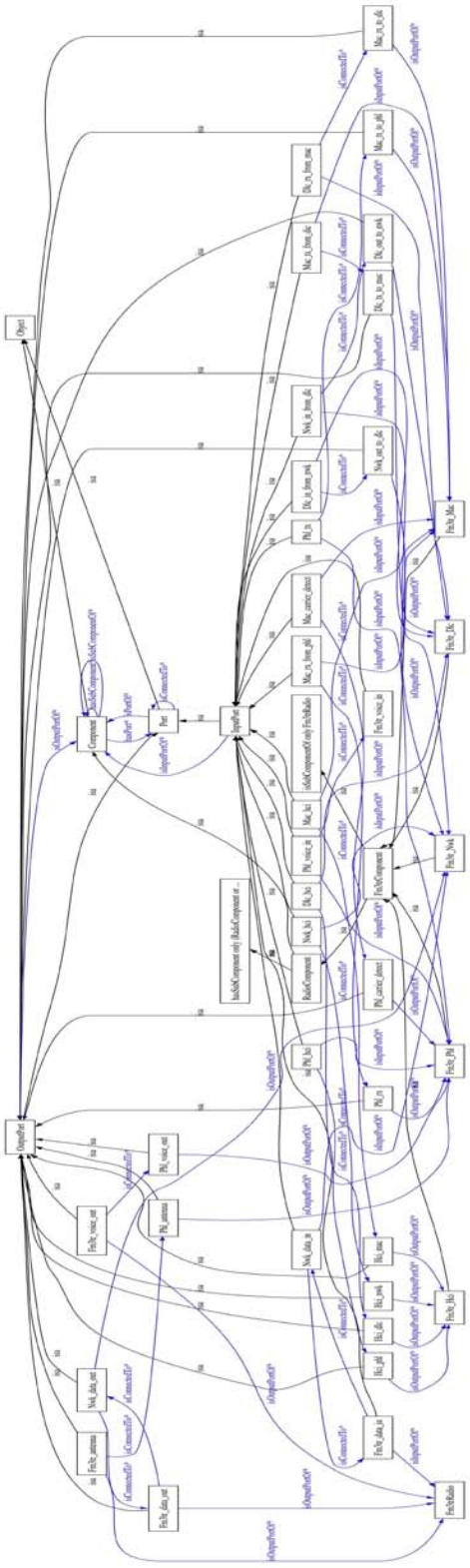

Figure 28 OWL Representation of FM3TR Radio

3.10.3 Behavior of Component

The behavior of a radio component is usually described by a behavior model, e.g. PetriNet or State Transition Diagram (STD). Figure 29 shows the structure and the behavior model of an FM3TR Physical Layer component. The representation of *BehaviorModel* can be found in Section 4.11. The relationship between a component and its behavior model is shown in Figure 30.

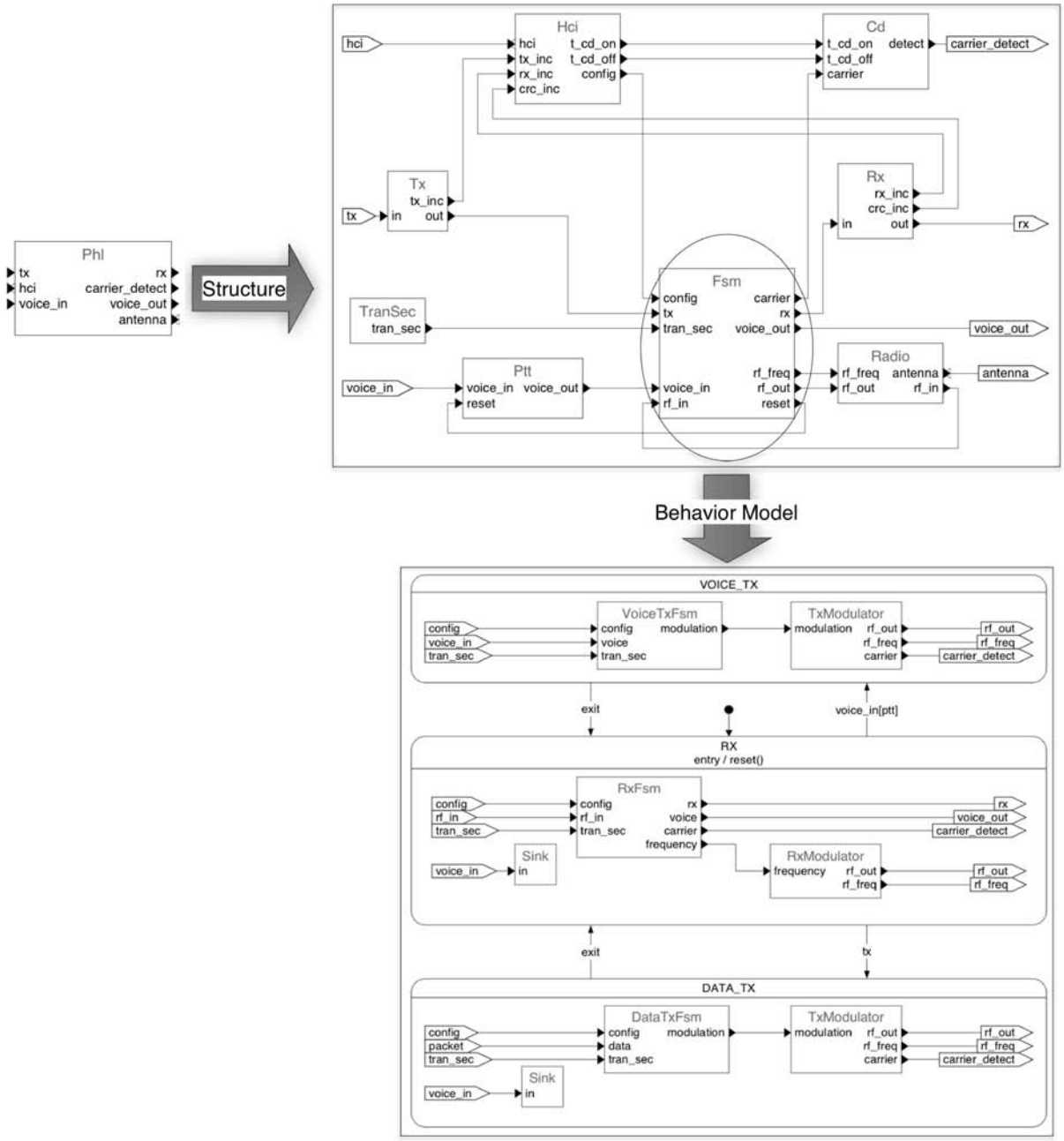

Figure 29 Structures and Behavior Model of FM3TR Physical Layer Component (Source: [9])

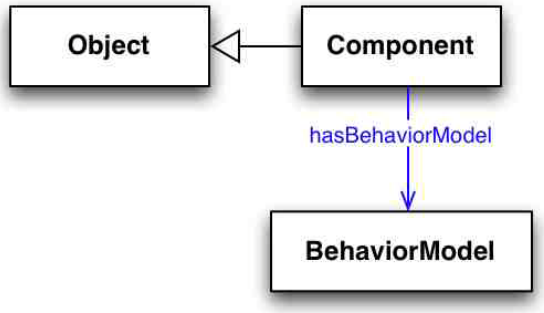

Figure 30 Relationships between Component and Behavior Model

3.10.4 Capabilities of Component

A *Component* is capable of performing particular tasks, such as receiving signal or detecting spectrum opportunities. The capabilities of a *Component* are a set of processes. Therefore, we use an object-type property *hasCapability* to represent the capabilities of a *Component*, shown in Figure 31.

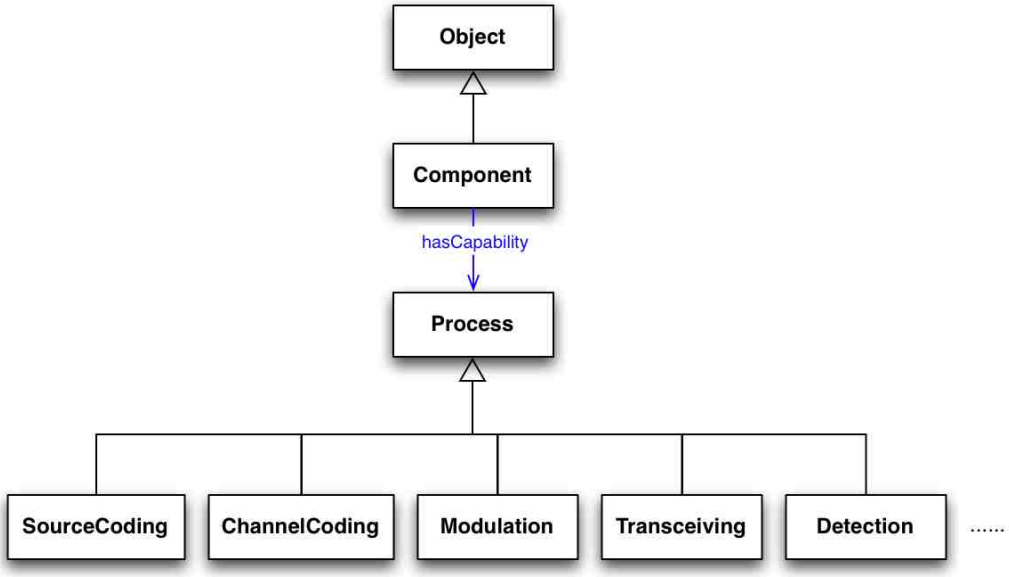

Figure 31 Capabilities of Component

3.10.5 API of Software Component

A physical component contains input and output ports. A software component can implement a set of APIs to enable the interaction with other software components. Once a software component implements an API, other software components can use this API.

The relationship between component and API is shown in Figure 32.

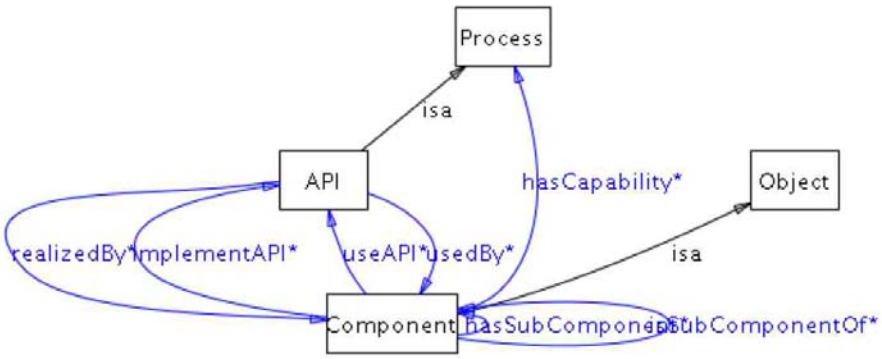

Figure 32 Relationships between Component and API

3.10.5.1 Example: OWL Representation of APIs of Transmitter

The API of Transmitter is specified in the "Transceiver Facility Specification" document by the Transceiver Working Group of Wireless Innovation Forum [10]. Figure 33 shows the API between Transmitter and Waveform Application. Figure 34 and Figure 35 show the overview of each API of Transmitter.

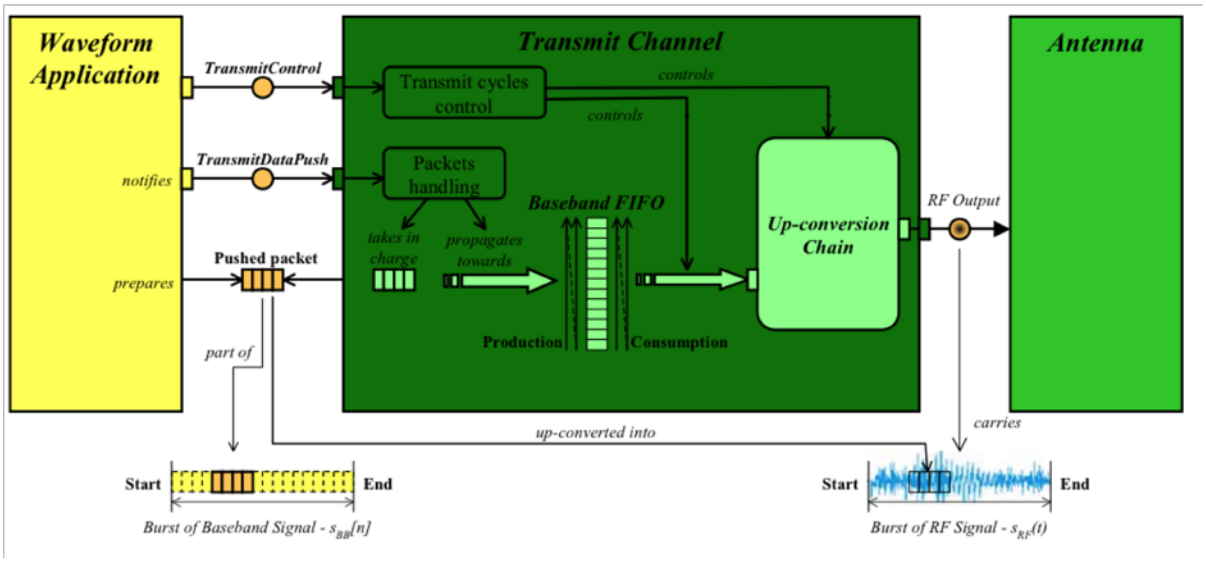

Figure 33 Overview of Transmitter API

Signature summary (pseudo-code)	Used by	Realized by	Description
createTransmitCycleProfile(Time requestedTransmitStartTime, Time requestedTransmitStopTime, UShort requestedPresetId, Frequency requestedCarrierFrequency, AnaloguePower requestedNominalRFPower)	Waveform Application	Transceiver Subsystem	Creation of a Transmit Cycle Profile.
configureTransmitCycle(Ulong targetCycleId, Time requestedTransmitStartTime, Time requestedTransmitStopTime, Frequency requestedCarrierFrequency, AnaloguePower requestedNominalRFPower)	Waveform Application	Transceiver Subsystem	Configuration of an existing Transmit Cycle Profile.
setTransmitStopTime(Ulong targetCycleId, Time requestedTransmitStopTime)	Waveform Application	Transceiver Subsystem	Specification of the end time of a Transmit Cycle.

Figure 34 Transmitter API (1): TransmitControl

Signature summary (pseudo-code)	Used by	Realized by	Description
pushBBSamplesTx(BBPacket thePushedPacket, Boolean endOfBurst)	Waveform Application	Transceiver Subsystem	Notifies availability of a baseband samples packet.

Figure 35 Transmitter API (2): TransmitDataPush

Figure 36 shows the OWL representation of the Transmitter API. The details of API and Method are shown in Section 2.3.3.2.

3.10.6 NetworkMembership of Component

A *Component* may have membership in a *Network.* Therefore each *Component* is associated with one or more than one *NetworkMembership*. The relationships among *Component*, *Network*, *NetworkMembership* and *Role* are shown in Section 3.13.

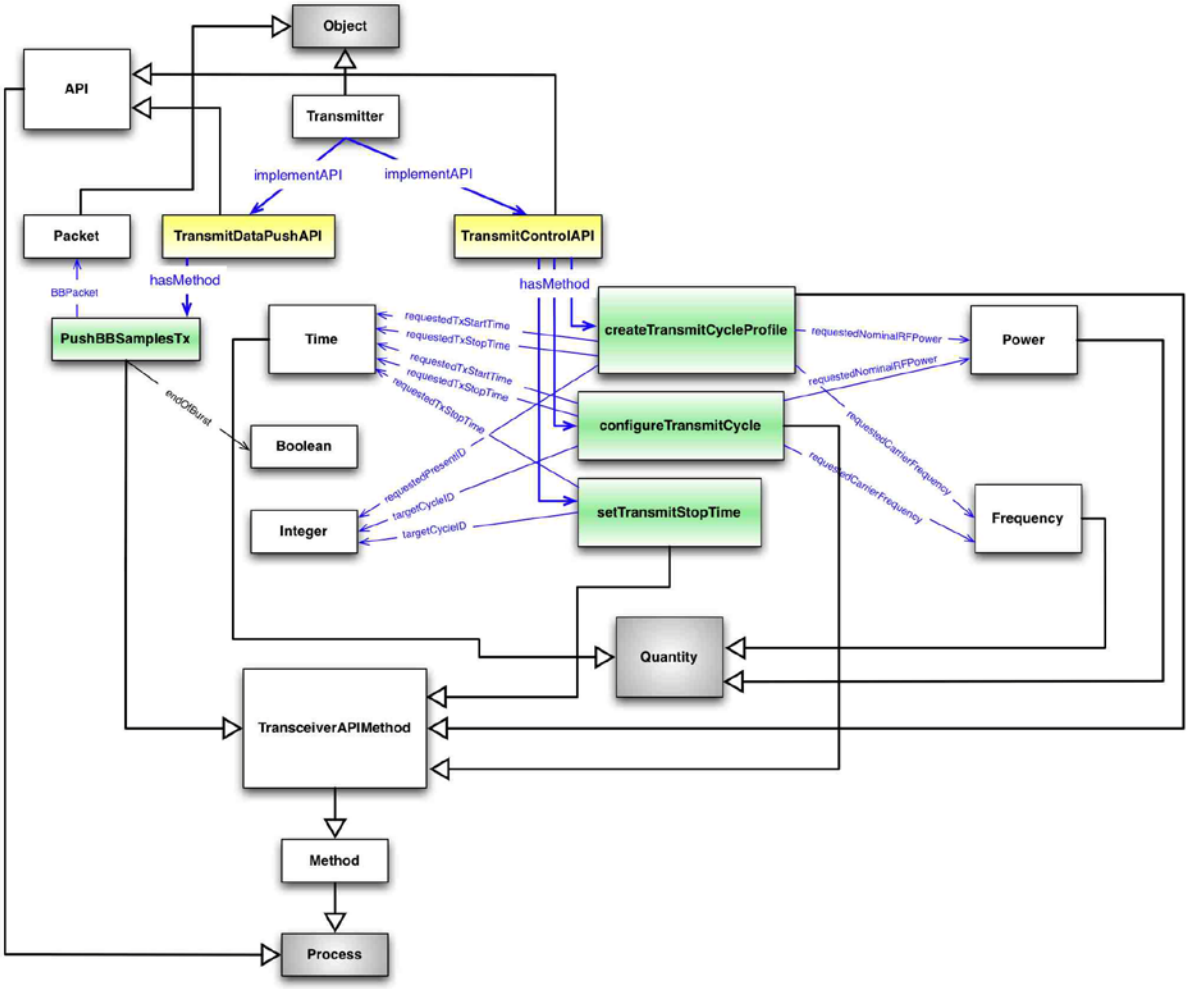

Figure 36 OWL Representation of Transmitter API

3.11 TransceiverPreset, Transfer Functions and Constraints of Transfer Functions

3.11.1 TransceiverPreset

TransceiverPreset refers to a set of tunable parameters that are provided with corresponding requested values before the up-/down-conversion is activated.

According to [10], *TransceiverPreset* is composed of the tunable parameters of *BasebandSampleSequence*, *ChannelMask*, *GroupDelayMask* and *SpectrumMask*.

3.11.2 TxChannelTransferFunction and RxChannelTransferFunction

TxChannelTransferFunction refers to the transfer function response of the transformation operated by up-conversion chain between the *BasebandSampleSequence* and *RFAnalogSignal*.

RxChannelTransferFunction refers to the transfer function response of the transformation operated by the down-conversion chain between the *RFAnalogSignal* and *BasebandSampleSequence*.

TxChannelTransferFunction is used to characterize the process of *Transmitting* and *RxChannelTransferFunction* is used to characterize the process of *Receiving*.

3.11.3 ChannelMask, SpectrumMask, and GroupDelayMask

ChannelMask, SpectrumMask and *GroupDelayMask* are the constraints of the *TxChannelTransferFunction* and *RxChannelTransferFunction* [10].

3.11.3.1 ChannelMask

ChannelMask refers to the requirements that shall be met by the *Channel Transfer Function* of a given conversion chain.

3.11.3.2 SpectrumMask

SpectrumMask is used to characterize the spectrum mask to be satisfied by the modulus of the *Channel Transfer Function*.

Figure 37 shows the characteristics of *SpectrumMask.*

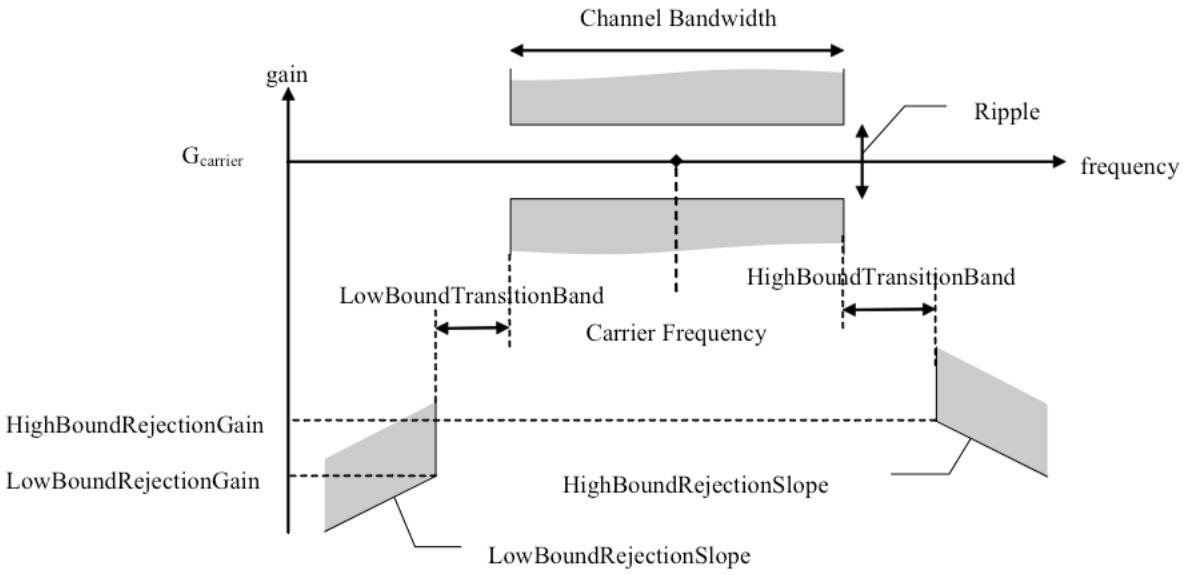

Figure 37 Characteristics of Spectrum Mask (Source: [10])

3.11.3.3 GroupDelayMask

GroupDelayMask is used to characterize the group delay response to be satisfied by the Channel Transfer Function. Figure 38 shows the characteristics of *GroupDelayMask*.

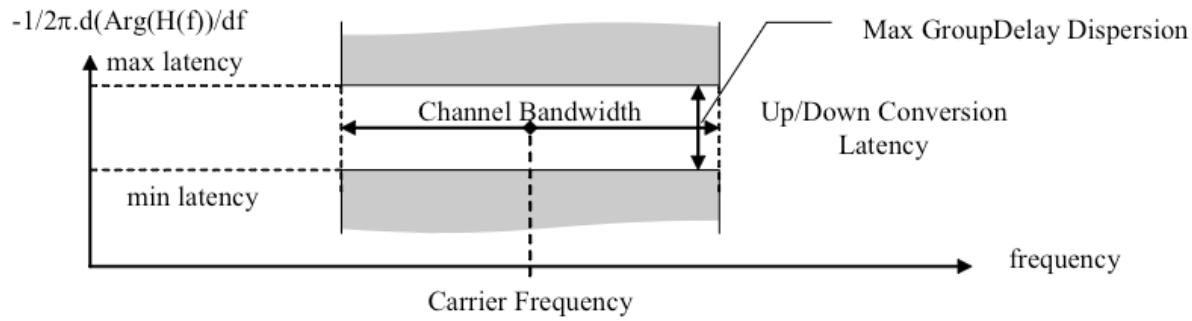

Figure 38 Characteristics of GroupDelayMask (Source: [10])

3.11.3.4 Properties of ChannelMask, SpectrumMask, and GroupDelayMask

Table 7 shows the overview of the properties associated to *ChannelMask*, *SpectrumMask* and *GroupDelayMask.*

Table 7 Properties of ChannelMask, SpectrumMask and GroupDelayMask

	Property	**Domain**	**Range**
ChannelMask	carrierFrequencyAccuracy	ChannelMask	Frequency
	channelBandwidth	ChannelMask	Frequency
SpectrumMask	highBoundRejectionGain	SpectrumMask	Decibel
	highBoundRejectionSlope	SpectrumMask	GainSlope
	highBoundTransitionBand	SpectrumMask	Frequency
	lowBoundRejectionGain	SpectrumMask	Decibel
	lowBoundRejectionSlope	SpectrumMask	GainSlope
	lowBoundTransitionBand	SpectrumMask	Frequency
	ripple	SpectrumMask	Decibel
GroupDelayMask	maxGroupDelayDispersion	GroupDelayMask	Frequency

3.11.4 Summary of Transmitter-related Classes

Figure 39 shows the relationships among the classes related to *Transmitter*. *TransmitterPreset* and *Transmitter* are objects; they both participate in the process of *Tuning* and *Transmitting.* The details of *Tuning* and *Transmitting* will be described in Section 4.3 and Section 4.4.

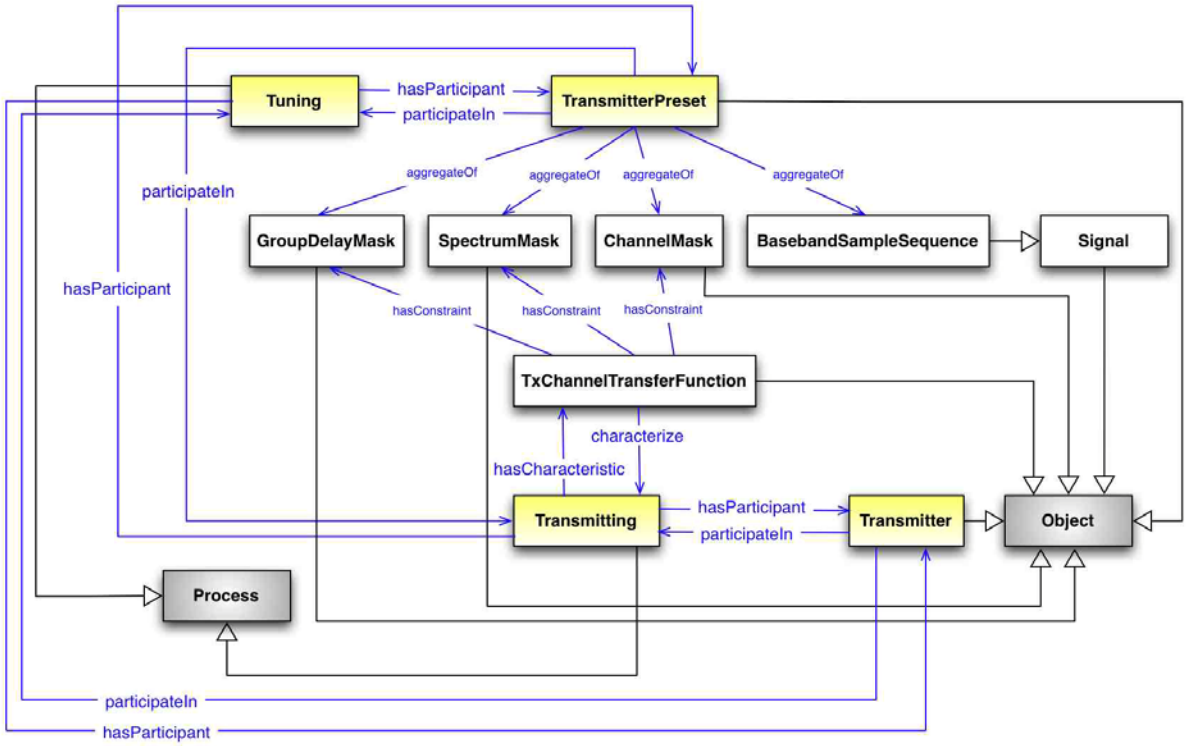

Figure 39 Characteristics of Transceiver

3.12 Detector and DetectionEvidence

A *Detector* is a device that can detect three types of *DetectionEvidence*: (1) *TimeEvidence*, (2) *LocationEvidence*, and (3) *SignalEvidence*.

The properties of *Detector* and its sub-classes are shown in Table 8.

Table 8 Properties of Detector and Its Sub-classes

	Property	Domain	Range
Detector	scanDuration	Detector	Time
	scanInterval	Detector	Time
	detectEvidence	Detector	DetectionEvidence
LocationDetector	detectEvidence	LocationDetector	LocationEvidence
SignalDetector	detectEvidence	SignalDetector	SignalEvidence
	endFrequency	SignalDetector	Frequency
	rssi	SignalDetector	Power
	sampleRate	SignalDetector	SampleRate
	setToDetect	SignalDetector	Signal
	signalDetectionPrecision	SignalDetector	Voltage
	signalDetectionThreshold	SignalDetector	Voltage
	signalToNoiseRatio	SignalDetector	Decibel
	startFrequency	SignalDetector	Frequency
TimeDetector	detectEvidence	TimeDetector	TimeEvidence

The relationship between *Detector* and *DetectionEvidence* is shown in Figure 40. The properties of *DetectionEvidence* and its sub-classes are shown in Table 9.

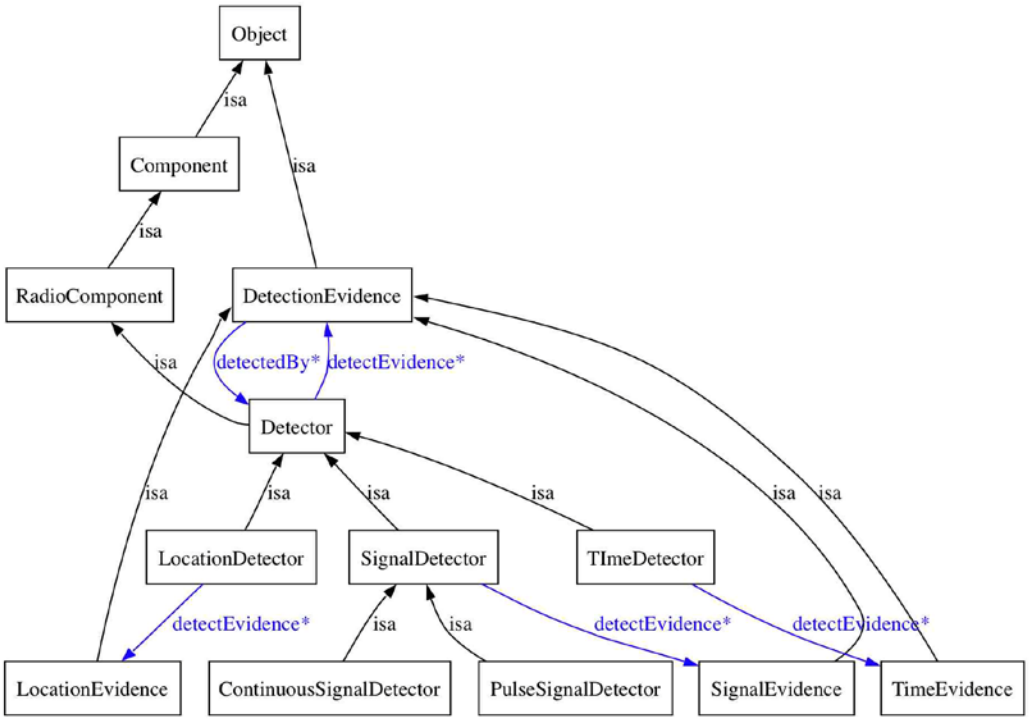

Figure 40 Relationships between Detector and DetectionEvidence

Table 9 Properties of DetectionEvidence and its Sub-classes

	Property	**Domain**	**Range**
Property of DetectionEvidence	confidence	DetectionEvidence	Percentage
	timeStamp	DetectionEvidence	Time
Property of LocationEvidence	location	LocationEvidence	Location
Property of TimeEvidence	time	TimeEvidence	Time
Property of SignalEvidence	consecutiveEmptyScanCount	SignalEvidence	Integer*
	detectedSignal	SignalEvidence	Signal
	lastCompleteEmptyScanDuration	SignalEvidence	Signal
	lastCompleteEmptyScanTime	SignalEvidence	Time
	lastDetectionTime	SignalEvidence	Time
	peakSensedPower	SignalEvidence	Power
	sensedEndFrequency	SignalEvidence	Frequency
	sensedStartFrequency	SignalEvidence	Frequency

3.13 Network, Network Membership and Role

A *Component* may act as a member in a *Network.* Each *NetworkMembership* is (1) associated with one *Component*, (2) belongs to a *Network*, and (3) has its *Role* in the *Network.* The *Role* of a member can be *master, slave* or *peer*. The relationships among *Network, NetworkMembership, Role* and *Component* are shown in Figure 41.

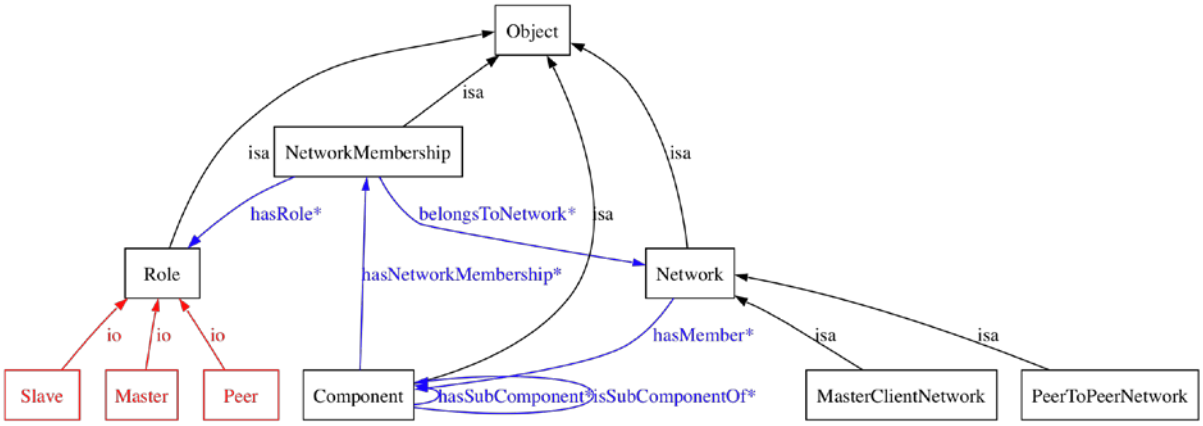

Figure 41 Relationships among Network, NetworkMembership, Role and Component

3.14 Agent and Goal

Agent is a special type of *Object*. The definition of *Agent* varies in different domains. In artificial intelligence, *Agent* refers to an autonomous entity which observes and acts upon an environment and directs its activity towards achieving its own goals [11]. The essences of an agent includes: (1) sensing of and reaction to the environment, i.e. an agent is able to sense the environment and react properly to the changes of the environment; (2) autonomy, i.e. an agent can perform a task without human intervention; (3) persistency, i.e., for example, if a software program is an agent, then it should be executed continuously over time rather than invoked on demand and terminates after the completion of its function; (4) goal-directed, an agent should be capable of choosing among multiple options and select the one that can achieve the goal [12].

The above properties distinguish an agent from an ordinary software program or module. In the domain of cognitive radio, a radio component has inputs and outputs. It performs tasks on its own by running a predefined algorithm. It could be said that the radio component senses the environment via the inputs and responds to the environment via outputs. In this sense, the radio component is capable of reacting to the environment and has some degree of autonomy. However, a radio component may be invoked for once and then goes into an idle state, waiting to be invoked again. In this sense, this radio component does not satisfy the temporal persistency property. Furthermore, in order to become an agent, a radio component must have goal-directed behavior, i.e. it does not simply sense and react upon the environment autonomously [12], it must be able to achieve a set of goals, e.g. avoid detection and interference, maximize throughput, etc.

DOLCE has a clear classification of *Agent*, i.e. which object is agentive and which is non-agentive. In this ontology, we do not restrict any of the radio components as a subclass of *Agent*. Instead, we define that an *Object* is an *Agent* if and only if it has a *Goal*. Given such a necessary and sufficient condition, it can be inferred whether a radio component is or is not an agent. The sub-classes of the *Goal* class and the relationships between *Agent* and *Goal* are shown in Figure 42.

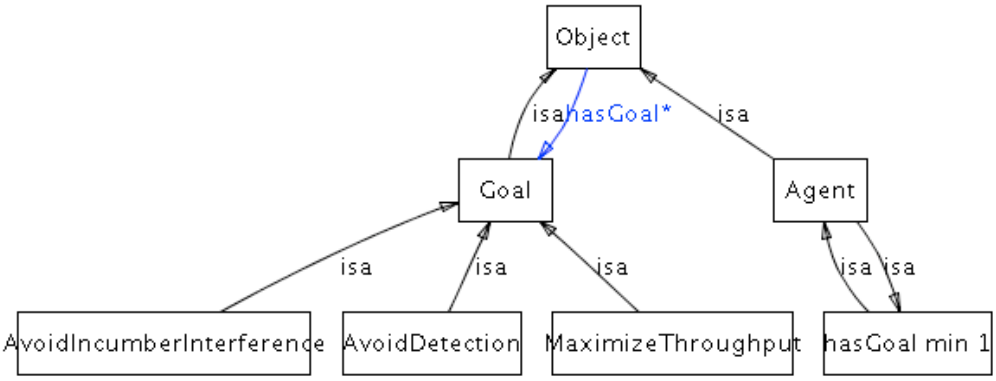

Figure 42 Relationships between Agent and Goal

4 Process

4.1 AIS and Protocol

Air Interface Specification (AIS) is closely related to the term Waveform. The Wireless Innovation Forum, working in cooperation with the IEEE Standards Association P1900.1 Working Group defines "waveform" as follows [3][13]:

a) The set of transformations and protocols applied to information that is transmitted over a channel and the corresponding set of transformations and protocols that convert received signals back to their information content.

b) The time-domain or frequency-domain representation of an RF signal.

c) The representation of transmitted RF signal plus optional additional radio functions up to and including all network layers.

AIS is the specification of a set of processes that are applied to the transmitted and received information. For instance, if two radios want to communicate with each other, the signals provided by the two radios must both satisfy the AIS, whereas the details of implementation may be different. In this sense, AIS is equivalent to the term Waveform defined in (a).

As it is discussed in Section 2.2.2.2, the specification of AIS is an *Object* whereas the implementation of AIS is a *Process.* In the current version, AIS refers to the implementation, thus it is a *Process.*

Typically, AIS is layered, with interfaces defined for each layer. Each layer consists of one or more protocols that perform the layer's functionality. A protocol defines the format and the ordering of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event.

For example, in cdma2000 1xEV-DO [14], the AIS is divided into several layers, shown in Figure 43. The protocols defined for each layer are shown in Figure 44. The MAC layer consists of multiple protocols such as *Control Channel Protocol* and *Forward Traffic Channel Protocol*. Hence, AIS is an aggregation of protocols. From another point of view, AIS is also an aggregation of various processes, i.e. AIS provides the specification for modulation, channel coding, source coding, etc. In our ontology, we only focus on the *physical layer*, *data link layer* and *network layer* of the AIS. The relationships among AIS, Protocol and Process are illustrated in Figure 45 .

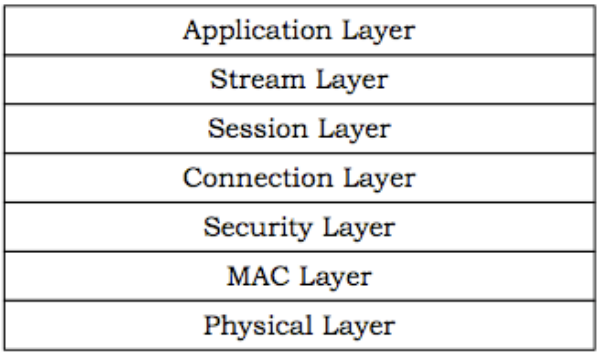

Figure 43 Air Interface Layering Architecture (Source: cdma2000 High Rate Packet Data Air Interface)

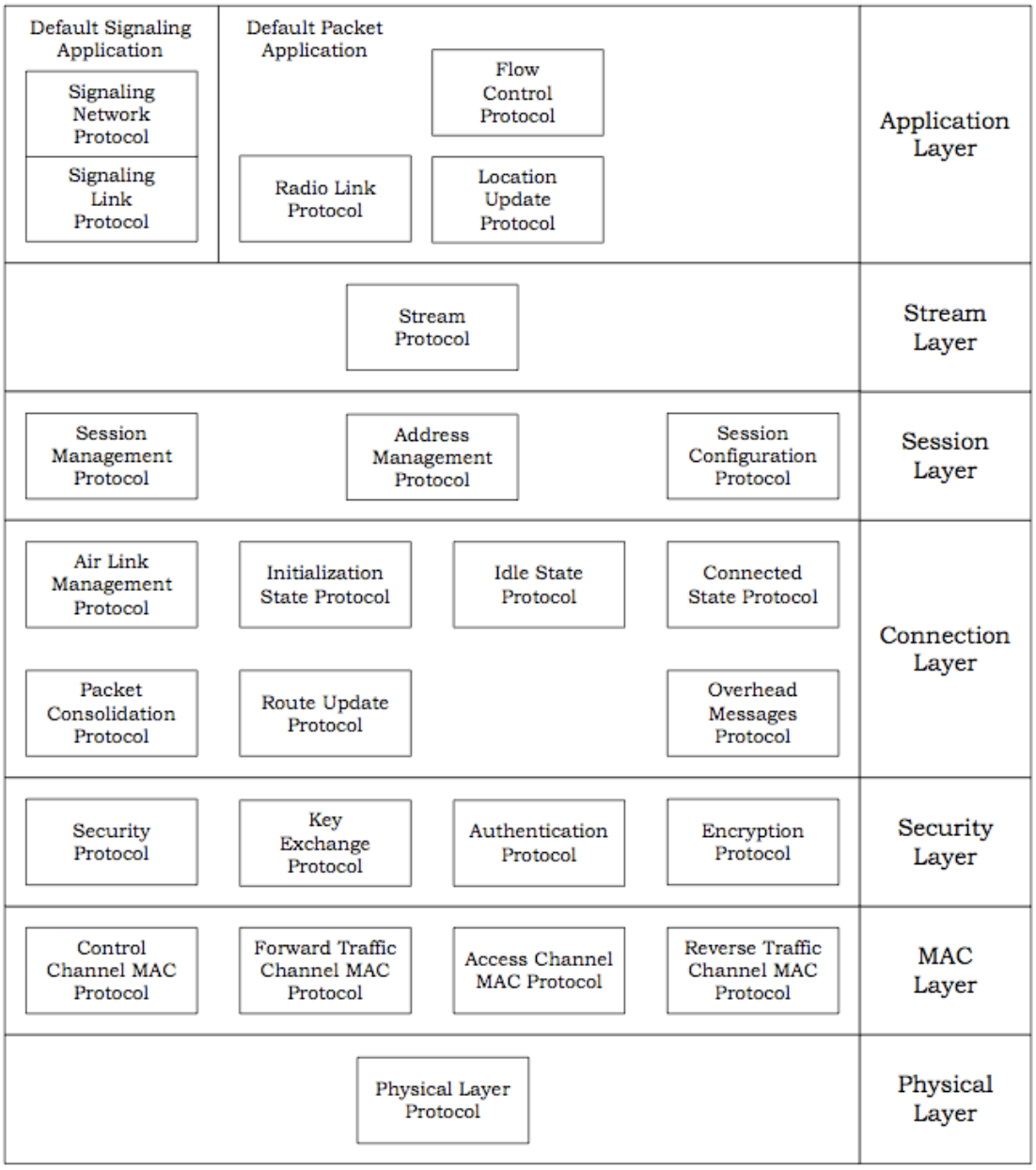

Figure 44 Default Protocols of cdma2000 1xEV-DO (Source: cdma2000 High Rate Packet Data Air Interface)

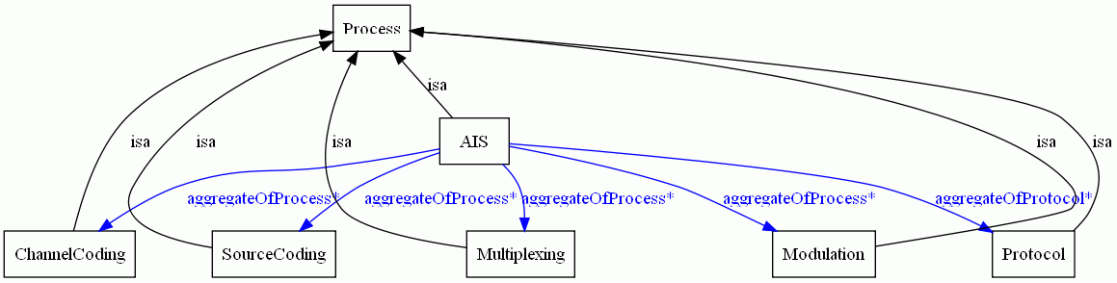

Figure 45 Relationships among AIS, Protocol and Process

4.2 API and Method

A general discussion of *API* and *Method* was already shown in Section 2.3.3.2 and Section 3.10.5. Here we provide some specific examples.

4.3 Tuning

Tuning refers to the process of setting the parameters of a radio to requested values. The relationships between *Tuning* and other classes related to *Transmitter* are shown in Section 3.11.4.

4.4 Transmitting

Transmitting refers to the process of up-converting bursts of *BasebandSampleSequence* to bursts of *RFAnalogSignal*. The *Transmitter* consumes the coming signal burst; stores the result in a buffer; then performs the up-conversion in real time.

Table 10 Properties of Transmitter

Property	Domain	Range
bandbandFIFOSize	Transmitter	Integer*
basebandCodingBits	Transmitter	Integer*
basebandNominalPower	Transmitter	Power
carrierFrequency	Transmitter	Frequency
consumptionStartTime	Transmitter	Time
consumptionStopTime	Transmitter	Time
maxCycleId	Transmitter	Integer*
maxOnTime	Transmitter	Time
maxPushedPacketSize	Transmitter	Integer*
maxTransmitDataPushInvocationDuration	Transmitter	Time
maxTuningDuration	Transmitter	Time
maxTxCycleProfiles	Transmitter	Integer*
maxUpconversionLatency	Transmitter	Time
minOffTime	Transmitter	Time
minPacketStorageAnticipation	Transmitter	Time
minReactivationTime	Transmitter	Time
minTransmitStartAnticipation	Transmitter	Time
minTransmitStartProximity	Transmitter	Time
nominalRFPower	Transmitter	Power
overflowMitigation	Transmitter	String*
reactivationTime	Transmitter	Time
transmissionPower	Transmitter	Power
transmitCycle	Transmitter	Integer*
transmitStartTime	Transmitter	Time
transmitStopTime	Transmitter	Time
transmitTimeProfileAccuracy	Transmitter	Time
tuningDuration	Transmitter	Time
tuningStartThreshold	Transmitter	Integer*
tuningStartTime	Transmitter	Time
upconversionLatency	Transmitter	Time

4.5 Receiving

Receiving refers to the process of down-converting bursts of *RFAnalogSignal* to bursts of *BasebandSampleSequence.*

4.6 SourceCoding (to be completed in future revision)

SourceCoding refers to the process of encoding information using fewer bits. Source coding helps reducing the consumption of hard disk or transmission bandwidth.

4.7 ChannelCoding (to be completed in future revision)

4.8 Modulation

The relationships of *Modulation*, *Modulator* and *Alphabet* were already discussed in Section 2.2.2.1.

In general, *Modulation* is a process that takes a digital signal as input and converts it to an analog signal. Then the analog signal is transmitted to the wireless channel. The changes in the carrier signal are chosen from a finite number of M alternative symbols, which is called *alphabet*.

4.9 Multiplexing (to be completed in future revision)

4.10 PNSequenceGeneration (to be completed in future revision)

PNSequenceGeneration refers to the process to generate the *PNCode*. *PNCode* can be used in scrambler and spectrum spreading.

4.11 BehaviorModel

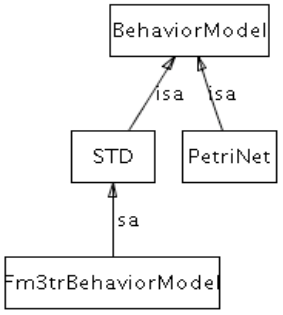

Figure 46 Sub-classes of BehaviorModel Class

The behavior of a radio component is usually described by a behavior model, e.g. *PetriNet* or *State Transition Diagram (STD)*. The FM3TR specification describes the behavior using State Transition Diagrams (STD). Thus, in our ontology, *FM3TRBehaviorModel* is a sub-class of STD. The subclasses of *BehaviorModel* are shown in Figure 46.

The basic elements of STD include (1) *State*, (2) *Transition* between states, (3) *Action* that is triggered by the state transition, (4) *Activity* that consists of a sequence of actions, (5) *Event* that triggers a state transition.

4.11.1 State

A state represents a stage in the behavior pattern of an object. It has three properties: (1) *doAction*, (2) *isFinal*, and (3) *isInitial*. *isFinal* and *isInitial* are the Boolean data type properties. The initial state is the state that an object is in when it is first created, whereas a final state is one in which no transitions lead out of. Also, a state transition will trigger an action, or a sequence of actions, thus a state is associated with an action by property *doAction.*

4.11.2 Transition

First, a *Transition* is a progression from one state to another. Thus, a *Transition* is associated with a target state and a source state. Second, as mentioned above, a state transition will trigger an *Action* or a sequence of actions. Therefore, a transition is associated with an action by property *cause.* Third, a *Transition* is usually triggered by an *Event* that is either internal or external to the object. Hence, a transition is associated with *Event* by property *causedBy.* Note that at this point, the distinctions among *Event*, *Condition* and *Guard* are not expressed in our ontology. In summary, the *Transition* class has four properties: (1) *cause*, (2) *casuedBy*, (3) *sourceState* and (4) *targetState.*

4.11.3 Action, Activity and Event

The distinction between *Action* and *Activity* is described in the UML superstructure specification [15]. In short, an *Action* is performed when a state transits to another state. An Activity consists of a sequence of *Actions*. Each *Action* in an *Activity* may execute zero, one, or more times for each activity execution. In our ontology, *Activity* is equivalent to *Process* and does not show up in the ontology as a separate class. The relationship between action and activity is modeled in the following way. First, *Action* is modeled as a sub-class of *Process*. Second, a *Process* is an aggregation of *Actions*. Third, an *Action* can be appended to another *Action*. A sequence of actions forms an *Activity* (*Process*).

In UML, an *Event* is a notable occurrence at a particular point in time. A state transition is triggered by an internal or external event.

4.11.4 State Transition Diagram (STD)

In this ontology, we only focus on the finite state machine (FSM) to represent behaviors. An FSM can be described using a state transition table, as shown in Figure 47. It can be seen that a *State Transition Diagram* can be viewed as an aggregation of *State* and *Transition*. The OWL representation of *STD* and the relationships among *STD*, *State*, *Transition*, *Event*, *Action*, and *Activity* (*Process*) are shown in Figure 48.

Current State → / Event that causes the transition ↓	State A	State B	State C
Event X	...	...	...
Event Y	...	State C	...
Event Z	...	...	...

Figure 47 State Transition Table

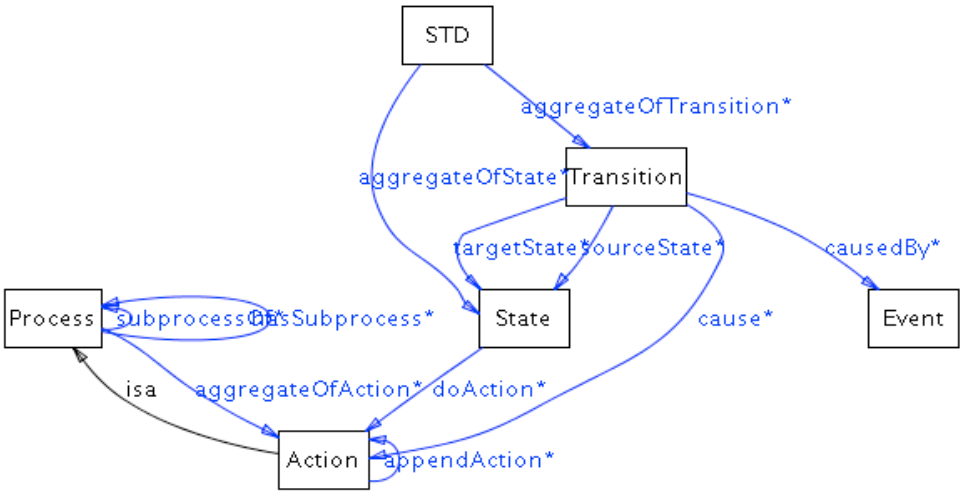

Figure 48 Representation of State Transition Diagram

4.11.5 Example: OWL Representation of Physical Layer FSM FM3TR Radio

In this section, an example from the FM3TR specification is used to illustrate how to use the approach described above to represent a Finite State Machine (FSM). Figure 49 shows a physical layer FSM specification of a FM3TR radio. This radio operates in a half-duplex mode. At the beginning, the radio idles at the RX state. When the PTT (push to talk) button is pressed and voice comes in from the Voice_Tx port, the radio transits from the RX state to the VOICE_Tx state. When it is finished, the radio will transit back to the RX state and reset the PTT. On the other hand, when there is data coming in from the TX port, the radio will transit from the Rx state to the DATA_TX state; when it is finished, the radio will transit back to the RX state and again reset the PTT. The diagram shows that each state specifies distinct receive and transmit activity. The realization of this FSM is shown in Figure 50.

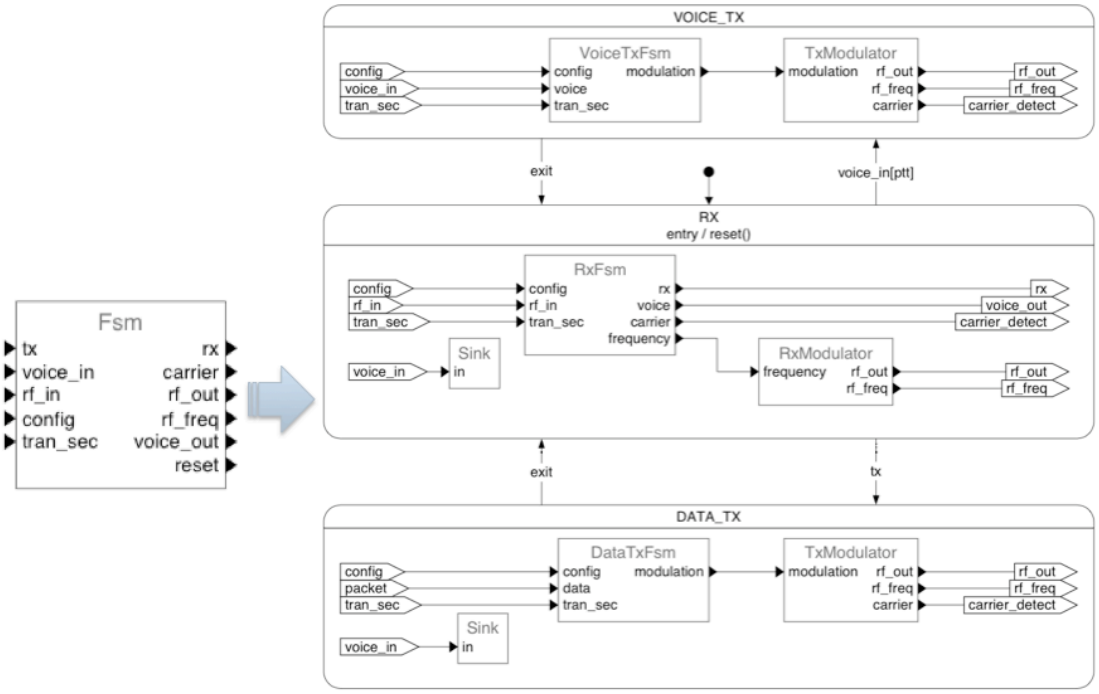

Figure 49 Physical Layer FSM Specification of a FM3TR Radio (Source: [9])

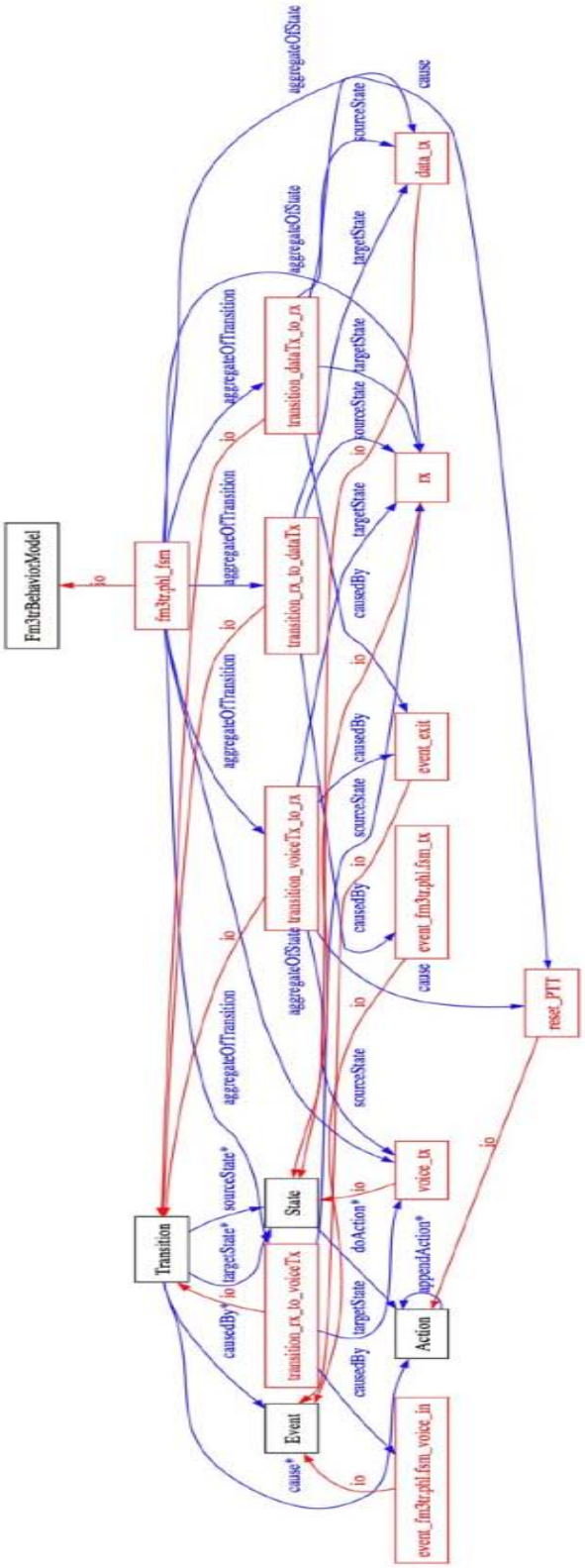

Figure 50 OWL Representation of the Physical Layer FSM of FM3TR Radio

5 Value

The subclasses of Value class include (1) CartesianCoordinates, (2) ComplexValue, (3) FloatValue, and (4) IntegerValue.

The properties of each subclass are shown in Figure 51.

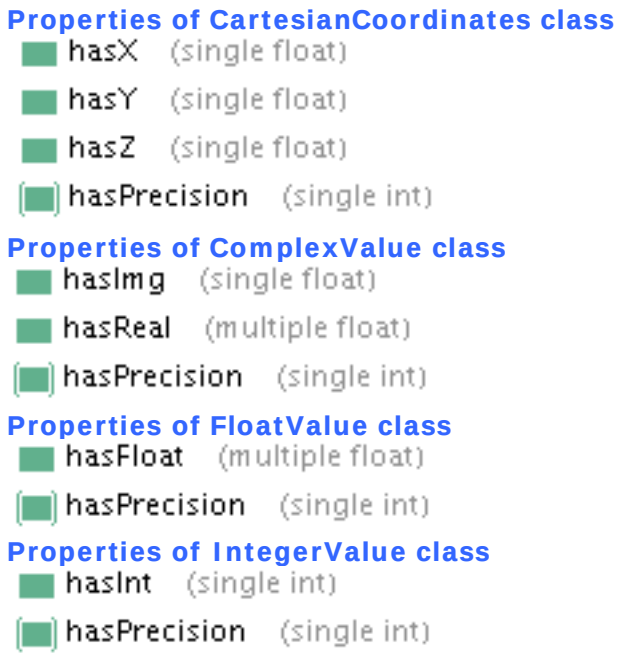

Figure 51 Properties of Value Class

6 Quantity and UnitOfMeasure

The summary of *Quanity* and *UnitOfMeasure* is shown in Table 11.

Table 11 Overviews of Quantity and UnitOfMeasure

Quantity			**UnitOfMeasure**
Bandwidth	ComputingBandwidth		Bits/sec
	SignalProcessBandwidth		Hz
Coordinates			
ElectricCurrent			I
ElectricFieldStrength			dBμV/m
Energy			joule
Frequency			Hz
Information			Bit
Length			Meter
Location	AreaLocation	LocationPolygone	
	LocationPoint	Latitude	
		Longitude	
		Altitude	
Power			dBm
PowerDensity			dBm/m^2 , mW/cm^2
Ratio	Decibel		
	Percentage		
SignalRate	ChipRate		Chip/sec
	Goodput		Bit/sec
	GrossBitRate		Bit/sec
	NetBitRate		Bit/Sec
	SampleRate		Sample/sec
	SymbolRate		Symbol/sec
	Throughput		Bit/Sec
Time			Second
Voltage			V

7 Summary and Future Work

In summary, the Cognitive Radio Ontology presented in this document has 230 classes and 188 properties, covering the basic terms of wireless communications from the PHY layer, MAC layer and Network layer. This ontology serves as the core ontology and is expected to be the foundation to the next step of the MLM project. In the future, we are planning to (1) develop policies and rules for policy based radio control based on the basic concepts defined in this ontology; (2) extend the core ontology and make it capable to express the use cases and support the policy based radio control.

8 References

[1] C. Masolo, S. Borgo, A. Gangemi, "DOLCE : a Descriptive Ontology for Linguistic and Cognitive Engineering". Technical report, Institute of Cognitive Science and Technology, Italian National Research Council, 2003.
[2] N. Markosian, "What Are Physical Objects?", Philosophy and Phenomenological Research, 61 (2000), pp. 375-395.
[3] IEEE Standard Definitions and Concepts for Dynamic Spectrum Access: Terminology Relating to Emerging Wireless Networks, System Functionality, and Spectrum Management, IEEE Std 1900.1, September 2008.
[4] http://www.agilemodeling.com/artifacts/classDiagram.htm
[5] R. Mizoguchi, "On Property: Property vs. Attribute", Technical Report, ISIR, Osaka University, [Online] Available: http://www.ei.sanken.osaka-u.ac.jp/main/documents/OnProperty.pdf.
[6]http://protege.stanford.edu/
[7] Shujun Li, Mieczyslaw M. Kokar, David Brady, "Developing an Ontology for the Cognitive Radio: Issues and Decisions", on Proceedings of the SDR'08 Technical Conference, December 2008, Available at http://groups.winnforum.org/p/cm/ld/fid=56
[8] http://en.wikipedia.org/wiki/Symbol_rate
[9] E. D. Willink, "Definition of Reactive Systems Using the Waveform Description Langauage", Proceedings of the SDR Forum/AFRL Waveform Development Environment Workshop, 1 November 2000
[10] "Transceiver Facility Specification, SDRF-08-S-0008-V1.0.0", Wireless Innovation Forum, Jan. 2009, Available at http://groups.winnforum.org/d/do/1554
[11] S. J. Russell, P. Norvig, "Artificial Intelligence: A Modern Approach (2nd ed.)", Prentice Hall, chapter. 2, 2003
[12] S. Franklin, A. Graesser, "Is it an agent, or just a program?", Intelligent Agents III, Springer, pp. 21–36, 1997.
[13] "Cognitive Radio Definitions", SDRF-06-R-0011-V1.0.0, Wireless Innovation Forum, June 2007, Available at http://groups.winnforum.org/d/do/1585
[14] "cdma2000 High Rate Packet Data Air Interface Specification", 3GPP2 C.S0024-A Version 2.0, [Online] Available: http://www.3gpp2.org/Public_html/specs/C.S0024-A_v2.0_050727.pdf
[15] UML Superstructure Specification version 2.1.2, [Online] Available: http://www.omg.org/spec/UML/2.1.2/