MiXiM - Physical Layer

Karl Wessel: wessel@tkn.tu-berlin.de
Michael Swigulski: swigulski@tkn.tu-berlin.de

04. October 2007

1 Preamble

1.1 What is the Physical Layer

TODO: write description

2 requirement specification

2.1 Overview

- provide status information to MAC
- switch states (RX, TX, SLEEP)
- send packets to air/channel
- receive packets / listen for packets
- provide hooks for statistical information
- configurable settings

2.2 provide status information to MAC

In addition to received packets the physical layer has to provide some other information to the MAC layer. Some of this information has to be provided passively on demand (e.g. current mode) and some should be delivered actively to the MAC layer on certain events (e.g. transmission of a packet complete). Information which has to be provided to MAC on demand:

- channelstate: idle (boolean) or RSSI
- current mode (RX, TX, SLEEP)

Information which has to be provided to MAC the moment it occurs:

- transmission over (send)

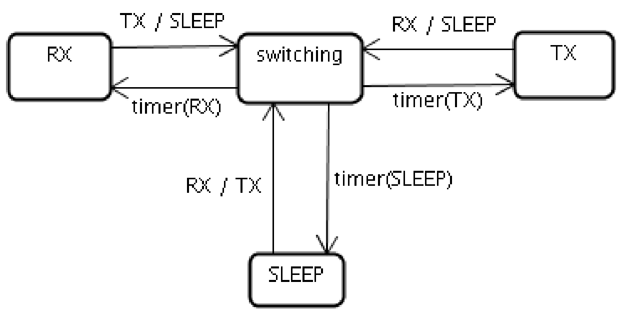


Figure 1: State machine for current mode

2.3 switch states

The physical layer has to be able to switch between the following things:

- current mode (RX, TX, SLEEP)

Switching from one mode to another may take some time. Whereas the switching time may depends to whitch mode we are switching.

2.4 send packets

The physical layer has to be able to send packets from the MAC Layer to the channel. We also want to support the possibility to control the sending process after it has been started..

Before we can send packets the following things have to be assured:

- the radio has to be in TX mode
- we are not already sending
- the channel should be idle (this is no hard requirement)

The above items should be controlled by the MAC layer so the physical layer would only throw an error if they are not set.

The following information has to be attached by the MAC layer to the packet:

- header bitrate
- payload bitrate
- channel
- TX power
- size of packet

The sending process itself is made up of the following steps:

1. MAC layer gives packet and control info to physical layer
2. check requirements for sending, throw error if they are not fulfilled
3. add information needed by the receiving physical layer to packet (see below)
4. add signal function transmission power[1] over time)[2] to packet
5. packet is send to channel by physical layer
6. schedule transmission over message for MAC layer

The following information is needed by the receiving physical layer:

- TX power (represented by the signal function)
- position, move direction and speed of the sending host
- the channel
- size of packet
- the duration the signal would need to be transmitted
- the duration of the preamble
- the bitrate (payload and header)

[1]The receiver converts the same signal to receiving power over time.

[2]The signal function could be more dimensional. E.g.: receiving power over time and channel.

2.5 receive packets

Because the packets arrive immediatly at every receiving node we have to simulate the receiving process:

1. simulate propagation delay (if needed)
2. simulate preamble duration
3. simulate payload duration

We also have to simulate the attenuation of the signal strength. This should be done by filtering the with the *analogue models*.

If the preamble is transmitted the packet has to be classified as *signal* or *noise*. The decision is made by the *decider*. Therefore the preamble has to be filtered previously by the analogue models.

If the transmission of a *signal* is over we have to decide if it was received correctly. This is also done by the *decider* by evaluating the *signal to noise ratio* short *SNR*. Of course we have to apply the *analogue model* to the signal and every noise interfering with the signal beforehand. If the signal was received correctly we pass it to the MAC Layer.

2.5.1 the analogue model

The *analogue model* simulates the attenuation of the signal strength by filtering the receiving power function.

There should be models to simulate the following things:

- pathloss
- shadowing
- fading

Further we set the following requirements to the *analogue models*:

- physical layer should be able to apply multiple *analogue models* to a signal
- you should be able to set the *analogue models* independent from physical layer
- you should be able to add your own *analogue models*

2.5.2 the decider

As mentioned already above the *decider* has to decide the following things:

- classify packet as signal or noise at base of the preamble
- decide if a packet was received correct at base of the signal at interfering noise

We set the following requirements to the *decider*:

- you should be able to set the *decider* independent from physical layer
- you should be able to add your own *decider*
- the *decider* should be able to return bitwise correctness of the *signal* (on demand)

2.6 statistical information

You should be able to get the following statistical information (the physical layer should not evaluate them but has to provide access to the according information):

- packet count
- received signal strength
- signal to noise ratio
- bit error ratio
- collisions

2.7 parameters

The following parameters of the physical layer should be freely configurable:

- simulate propagation delay? (boolean)
- which analogue models should be used
- the parameters for the analogue models
- which decider should be used
- the parameters for the decider
- thermal noise
- sensitivity
- maximum TX power
- switching times between modes (RX, TX, SLEEP)

2.8 list of requirements

1. provide status information to MAC
   (a) provide passive (on demand) **(see 2.2)**
   (b) provide active (messages) **(see 2.2)**
   (c) channelstate: idle (boolean) or RSSI **(see 2.2)**
   (d) current mode (RX, TX, SLEEP) **(see 2.2)**
   (e) transmission over event (send) **(see 2.2)** and **(see 2.4)**
2. switch states (RX, TX, SLEEP)
   (a) current mode **(see 2.3)**
   (b) switching times **(see 2.3)**
3. send packets to air/channel

   (a) get packet from MAC layer **(see 2.4)**
   (b) control sending process **(see 2.4)**
   (c) control information needed from MAC
      i. header bitrate **(see 2.4)**
      ii. payload bitrate **(see 2.4)**
      iii. channel **(see 2.4)**
      iv. TX power **(see 2.4)**
      v. size of packet **(see 2.4)**
   (d) prerequirements
      i. are in TX mode **(see 2.4)**
      ii. not already sending **(see 2.4)**
      iii. channel is idle **(see 2.4)**
   (e) attach informations for receiver
      i. attach transmission power over time function **(see 2.4)**
      ii. position, move direction and speed of the sending host **(see 2.4)**
      iii. channel **(see 2.4)**
      iv. size of packet **(see 2.4)**
      v. duration **(see 2.4)**
      vi. duration of preamble **(see 2.4)**
      vii. bitrate (payload and header) **(see 2.4)**
   (f) send to channel **(see 2.4)**

4. receive packets / listen for packets
   (a) simulate propagation delay **(see 2.5)**
   (b) simulate preamble **(see 2.5)**
   (c) simulate transmission duration **(see 2.5)**
   (d) simulate attenuation **(see 2.5)**
   (e) filter preamble **(see 2.5)**
   (f) filter signal and interfering noise **(see 2.5)**
   (g) pass correct packets to MAC **(see 2.5)**
   (h) analogue model
      i. filter signal strength **(see 2.5.1)**
      ii. simulate pathloss **(see 2.5.1)**
      iii. simulate shadowing **(see 2.5.1)**
      iv. simulate fading **(see 2.5.1)**
      v. more than one analogue model per phy **(see 2.5.1)**
      vi. can be set independent from phy **(see 2.5.1)**
      vii. can add own analogue models **(see 2.5.1)**

5. decider
   (a) classify preamble as noise or signal **(see 2.5)** and **(see 2.5.2)**

   (b) decide if packet was received correct **(see 2.5)**
   (c) can be set independent from phy **(see 2.5.2)**
   (d) can add own decider **(see 2.5.2)**
   (e) return bitwise errors **(see 2.5.2)**

6. provide hooks for statistical information
   (a) packet count **(see 2.6)**
   (b) received signal strength**(see 2.6)**
   (c) signal to noise ratio **(see 2.6)**
   (d) bit error ratio **(see 2.6)**
   (e) collisions **(see 2.6)**

7. configurable settings
   (a) simulate propagation delay? (boolean) **(see 2.7)**
   (b) which analogue models should be used **(see 2.7)**
   (c) the parameters for the analogue models **(see 2.7)**
   (d) which decider should be used **(see 2.7)**
   (e) the parameters for the decider **(see 2.7)**
   (f) thermal noise **(see 2.7)**
   (g) sensitivity **(see 2.7)**
   (h) maximum TX power **(see 2.7)**
   (i) switching times between modes (RX, TX, SLEEP) **(see 2.7)**

3 modelling

Note: We denote a Layer in a general meaning by 'Phy-Layer' or 'MAC-Layer' and our concrete C++ classes by *BasePhyLayer* or *BaseMacLayer*.

3.1 overview

Here we present the design- and interface details of the OMNeT-module *BasePhyLayer* to meet the requirement specification. That includes:

1. internal class diagram of *BasePhyLayer* and relation to *BaseMacLayer*
2. interface description for all involved C++ classes
3. flow charts for reception of MacPacket from upper layer and AirFrame from the channel
4. some detailed flow charts for important sub processes

3.2 classgraph

We start with the classgraph for the OMNeT-module *BasePhyLayer* that shows its C++ classes, relations to other OMNeT-modules (especially *BaseMacLayer*) and the OMNeT-messages sent between them.

The *BasePhyLayer* holds a list(req. 4(h)v) of AnalogueModels and a pointer to a Decider. Thus the AnalogueModel and the Decider are submodules of *BasePhyLayer*. This way one is able to change(req. 4(h)vii)(req. 5d) and replace(req. 4(h)vi)(req. 5c) them independently from the *BasePhyLayer*.

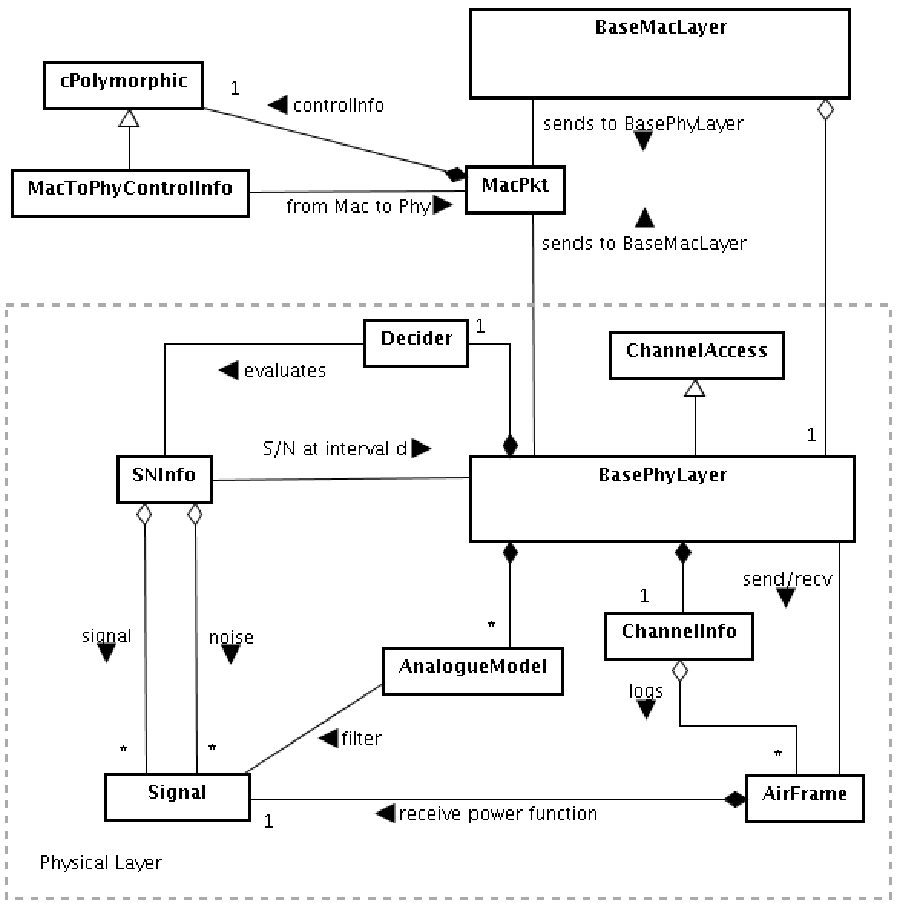


Figure 2: class graph

3.3 The *BasePhyLayer* interface

In this section we focus on how one is able to communicate with the *BasePhyLayer*, i.e. especially the *BaseMacLayer* which is connected to the *BasePhyLayer* in three ways:

1. OMNet-channel for data messages
2. OMNet-channel for control messages
3. a reference to *BasePhyLayer*.

The data channel is used to send and receive$^{(\text{req. 3a})}$ MacPkts to and from the *BasePhyLayer*.

The control channel is used by the *BasePhyLayer* to inform the *BaseMacLayer* about certain events$^{(\text{req. 1b})}$, like the TX_OVER$^{(\text{req. 1e})}$ message which indicates the end of a sending transmission.

The reference provides a passive way$^{(\text{req. 1a})}$ for the *BaseMacLayer* to get information about the current channel state$^{(\text{req. 1c})}$ and to get$^{(\text{req. 1d})}$ and set$^{(\text{req. 2a})}$ the current mode (RX, TX, SLEEP). Switching times$^{(\text{req. 2b})}$ from one mode to another are controlled internaly by a state machine. *see also Figure 1.*

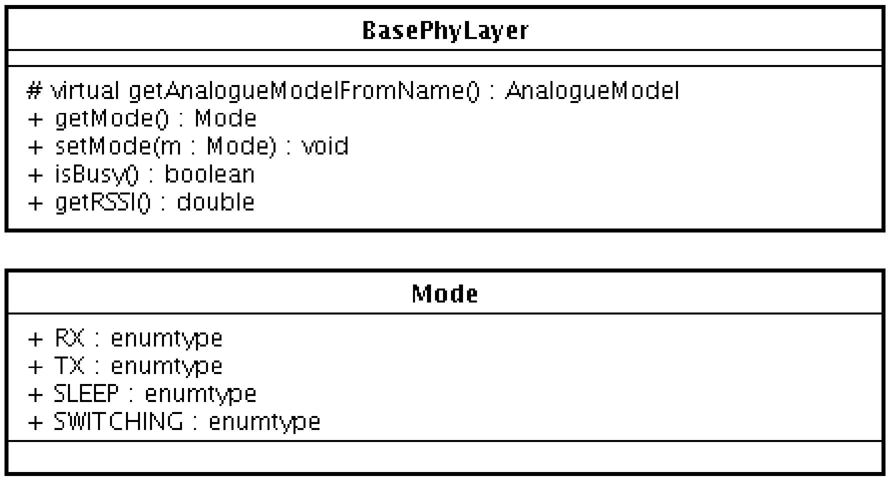


Figure 3: BasePhyLayer interface

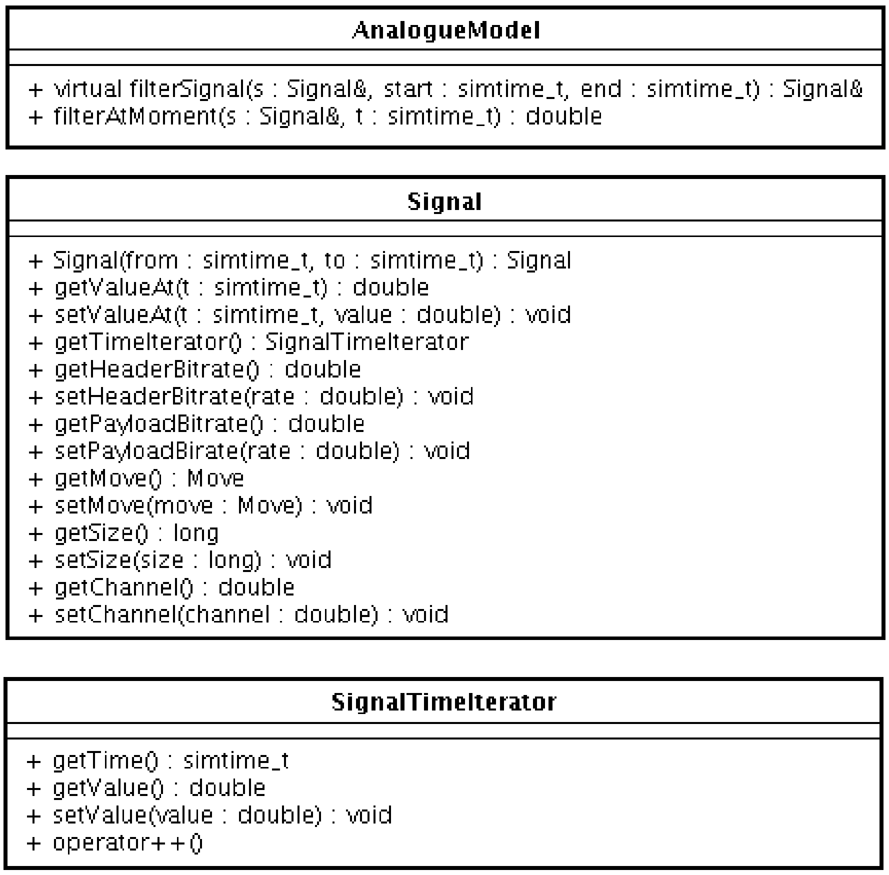
3.4 AnalogueModel and Signal

The Signal is designed one-dimensional (power-over-time) by default with a specified time point for start and end of the Signal. The owner is able to add and request values at a specific time point$^{(\text{req. 3(e)i})}$. The Method getTimeIterator() returns an appropriate SignalTimeIterator needed for applying AnalogueModels to the Signal.

NOTE: Anyone who subclasses Signal should make shure to have a properly working SignalTimeIterator (subclassed) for it. The Signal-TimeIterator should always iterate over every time stamp in each dimension. This way simple AnalogueModels will be able to filter the Signal independent from its dimension.

Further the Signal is set the packets header and payload bitrate$^{(\text{req. 3(e)vii})}$, the Move of the Host$^{(\text{req. 3(e)ii})}$, the size of the packet$^{(\text{req. 3(e)iv})}$ and the channel to send the packet to$^{(\text{req. 3(e)iii})}$ by *BasePhyLayer*.

See also 3.7.

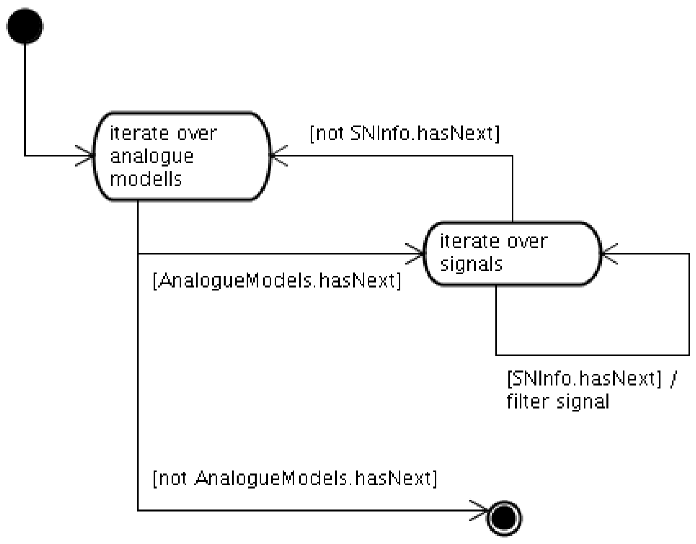


```
AnalogueModel

+ virtual filterSignal(s : Signal&, start : simtime_t, end : simtime_t) : Signal&
+ filterAtMoment(s : Signal&, t : simtime_t) : double

Signal

+ Signal(from : simtime_t, to : simtime_t) : Signal
+ getValueAt(t : simtime_t) : double
+ setValueAt(t : simtime_t, value : double) : void
+ getTimeIterator() : SignalTimeIterator
+ getHeaderBitrate() : double
+ setHeaderBitrate(rate : double) : void
+ getPayloadBitrate() : double
+ setPayloadBirate(rate : double) : void
+ getMove() : Move
+ setMove(move : Move) : void
+ getSize() : long
+ setSize(size : long) : void
+ getChannel() : double
+ setChannel(channel : double) : void

SignalTimeIterator

+ getTime() : simtime_t
+ getValue() : double
+ setValue(value : double) : void
+ operator++()
```

Figure 4: analogue model interface

The AnalogueModel offers functionality to filter a referenced signal$^{(\text{req. 4(h)i})}$ in a specified interval$^{(\text{req. 4f})}$ (e.g. preamble$^{(\text{req. 4e})}$) or at a single point in time.

Three basic AnalogueModel classes are foreseen to be plugged into Phy-Layer to simulate pathloss$^{(\text{req. 4(h)ii})}$, shadowing$^{(\text{req. 4(h)iii})}$ and fading$^{(\text{req. 4(h)iv})}$. *BasePhyLayer* is designed to apply an arbitrary number of AnalogueModels to a Signal.

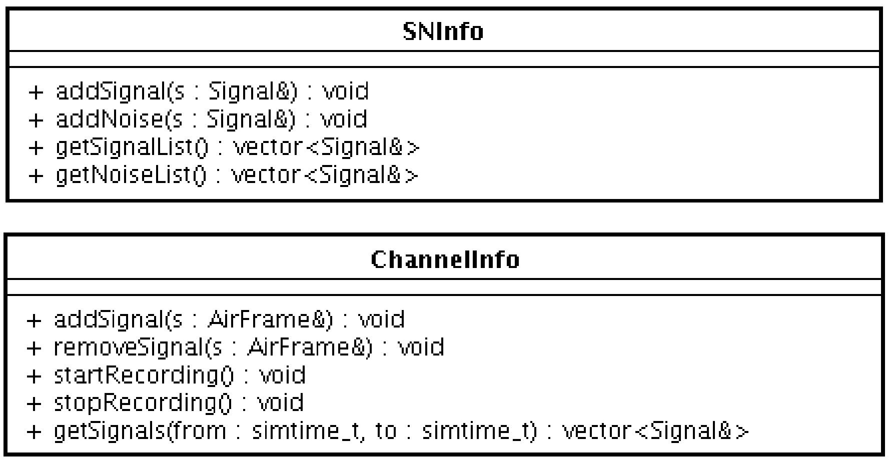


iterate over analogue modells
[not SNInfo.hasNext]
iterate over signals
[AnalogueModels.hasNext]
[SNInfo.hasNext] / filter signal
[not AnalogueModels.hasNext]

Figure 5: application of analogue models

3.5 SNInfo and ChannelInfo

ChannelInfo keeps track of all AirFrames on the channel. It does not differentiate between *signal* and *noise*. *BasePhyLayer* is able to add and remove references to certain AirFrames to and from ChannelInfo.
ChannelInfo is able to record the whole channel over time from a start to a stop signal and can return a vector of Signals (references) that intersect with a given time interval.
SNInfo is created by *BasePhyLayer* when a packet arrives to collect all signals from the channel that intersect with the reception time interval of the packet.



SNInfo
+ addSignal(s : Signal&) : void
+ addNoise(s : Signal&) : void
+ getSignalList() : vector<Signal&>
+ getNoiseList() : vector<Signal&>
ChannelInfo
+ addSignal(s : AirFrame&) : void
+ removeSignal(s : AirFrame&) : void
+ startRecording() : void
+ stopRecording() : void
+ getSignals(from : simtime_t, to : simtime_t) : vector<Signal&>

Figure 6: channel details

3.6 Decider

The Decider has two tasks:

1. It decides whether we are able to receive a certain packet by evaluting the SNInfo for the packets preamble time interval, otherwise the packet will be considered noise$^{(\text{req. 5a})}$

2. When a packet has been received and is not noise the Decider returns a DeciderResult for that packet, that only contains if the packet was received correct or not correct by default$^{(\text{req. 5b})}$.

A Decider that gives a richer DeciderResult (e.g. bitwise errors$^{(\text{req. 5e})}$) must be subclassed and implemented by the user.

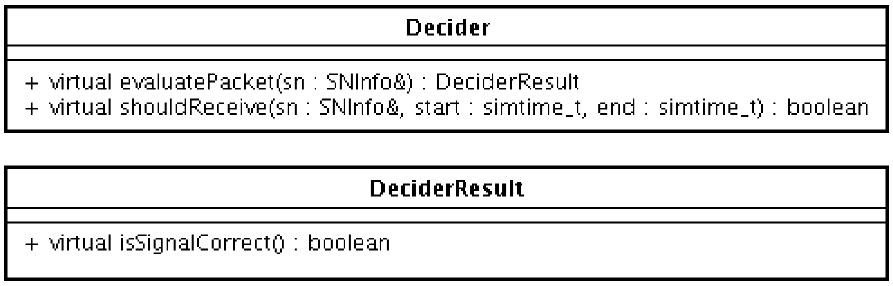
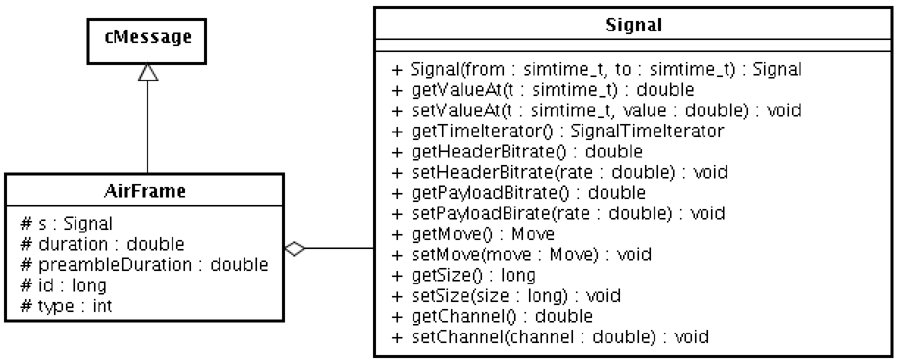

Decider
+ virtual evaluatePacket(sn : SNInfo&) : DeciderResult
+ virtual shouldReceive(sn : SNInfo&, start : simtime_t, end : simtime_t) : boolean
DeciderResult
+ virtual isSignalCorrect() : boolean

Figure 7: Decider interface

3.7 AirFrame

AirFrame and Signal are both constructed by *BasePhyLayer* with the help of MacToPhyControlInfo. It is shown below how nessecary information for sending/receiving is distributed. Signal is already discussed in 3.4.

BasePhyLayer calculates duration$^{(\text{req. 3(e)v})}$ and preamble duration$^{(\text{req. 3(e)vi})}$ of the packet and adds it to the AirFrame. To be able to control the sending process$^{(\text{req. 3b})}$ of another AirFrame every AirFrame has a unique id and a specific type which secifies if this is a normal AirFrame or a control-AirFrame.



cMessage
AirFrame
# s : Signal
# duration : double
# preambleDuration : double
# id : long
# type : int
Signal
+ Signal(from : simtime_t, to : simtime_t) : Signal
+ getValueAt(t : simtime_t) : double
+ setValueAt(t : simtime_t, value : double) : void
+ getTimeIterator() : SignalTimeIterator
+ getHeaderBitrate() : double
+ setHeaderBitrate(rate : double) : void
+ getPayloadBitrate() : double
+ setPayloadBirate(rate : double) : void
+ getMove() : Move
+ setMove(move : Move) : void
+ getSize() : long
+ setSize(size : long) : void
+ getChannel() : double
+ setChannel(channel : double) : void

Figure 8: member arrangement in AirFrame and Signal

3.8 receiving a MacPkt

On reception of a MacPkt from the MAC-Layer, *BasePhyLayer* checks if:

1. the radio is in TX mode$^{(\text{req. } 3(d)i)}$,
2. it is not already sending a packet$^{(\text{req. } 3(d)ii)}$ and
3. the channel is idle$^{(\text{req. } 3(d)iii)}$ (this is no hard requirement, *BasePhyLayer* could send anyway).

If condition 1 or 2 is not fulfilled it will throw an error.

The MacToPhyControlInfo object attached to the MacPkt contains the information needed by *BasePhyLayer* when constructing Signal and AirFrame to send to the channel. Right now it contains:

1. the channel for sending$^{(\text{req. } 3(c)iii)}$,
2. header bitrate$^{(\text{req. } 3(c)i)}$,
3. payload bitrate$^{(\text{req. } 3(c)ii)}$,
4. TX Power$^{(\text{req. } 3(c)iv)}$ and
5. the size of the packet$^{(\text{req. } 3(c)v)}$.

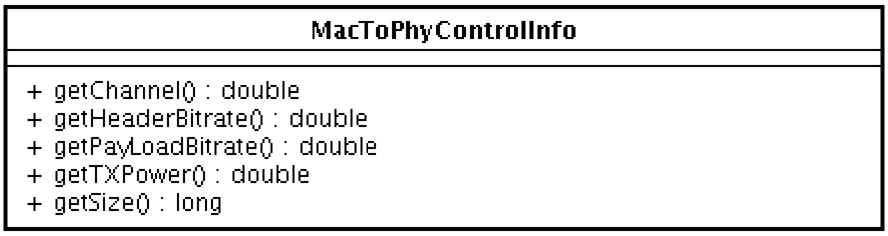


Figure 9: MacToPhyControlInfo interface

BasePhyLayer is responsilbe for creating AirFrame and Signal and attaching information (parameters) to them. For detailed arrangement of information in Signal and AirFrame see 3.7. When the AirFrame is complete and sent, *BasePhyLayer* schedules a TX_OVER message to the *BaseMacLayer*(via control-message).

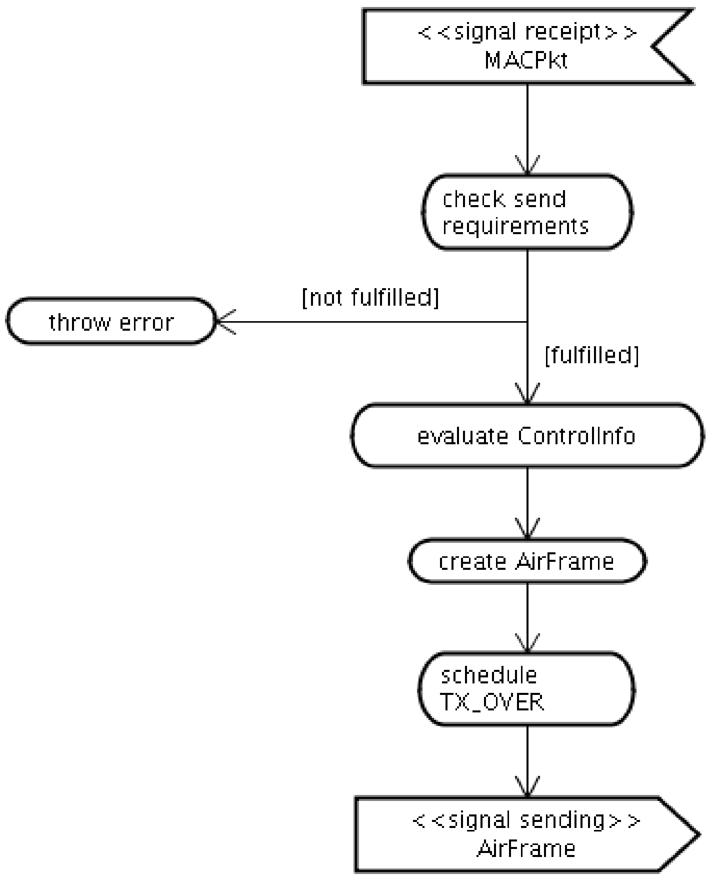


Figure 10: sending process

3.9 Receiving and processing an AirFrame

The reception of an AirFrame is divided into:

1. optional propagation delay$^{(\text{req. 4a})}$,
2. reception of the preamble$^{(\text{req. 4b})}$,
3. application of AnalogueModels to the corresponding SNInfo$^{(\text{req. 4d})}$,
4. decision whether packet is considered noise (Decider),
5. reception of the packet$^{(\text{req. 4c})}$.

Afterwards the packet is either dropped (if considered noise) or processed.

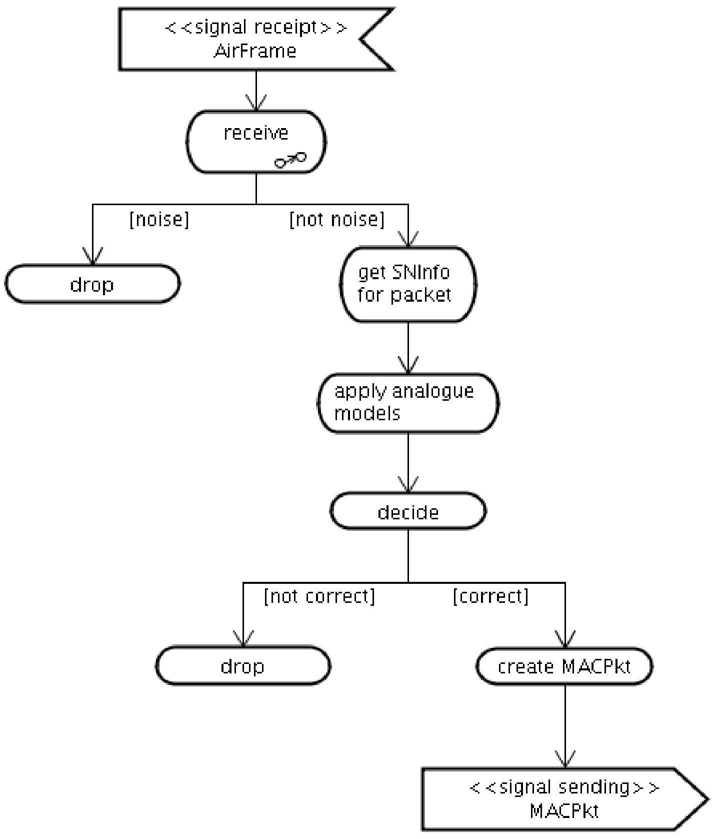


Figure 11: receiving process

The receiving process is modelled internally by a state machine that schedules the AirFrame that is received (since we have a pointer to it from the beginning) everytime a delay/time interval shall be simulated. That saves us the creation of additional self-messages.

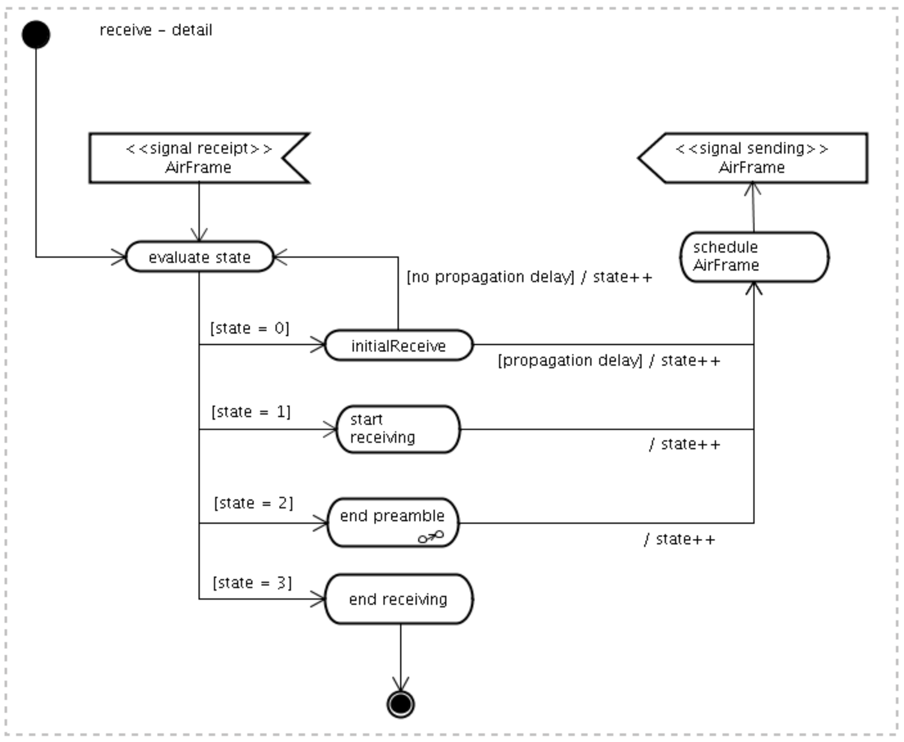


Figure 12: receive detail

When the preamble of a packet is completely received, *BasePhyLayer* constructs a SNInfo for the preamble, applies the AnalogueModels to it and passes it to the Decider to find out whether this packet is considered noise.

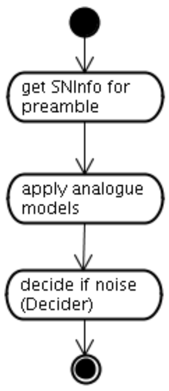


Figure 13: end preamble detail

In case a received packet is not *noise* it is processed, i.e. *BasePhyLayer* creates the corresponding SNInfo for the packet, applies AnalogueModels to it and passes the result to the Decider to check whether the packet was received correctly. If so, a MacPkt is created and handed up to Phy-Layer$^{\text{(req. 4g)}}$.

3.10 the .ned-file

The .ned-file of the *BasePhyLayer* has the following parameters:

- usePropagationDelay as *boolean*$^{\text{(req. 7a)}}$
- analogueModels as *XML*$^{\text{(req. 7b)(req. 7c)}}$
- decider as *XML*$^{\text{(req. 7d)(req. 7e)}}$
- thermalNoise as *numeric const*$^{\text{(req. 7f)}}$
- sensitivity as *numeric const*$^{\text{(req. 7g)}}$
- maximal TX power as *numeric const*$^{\text{(req. 7h)}}$
- switchTimeRX as *numeric const*$^{\text{(req. 7i)}}$
- switchTimeTX as *numeric const*$^{\text{(req. 7i)}}$
- switchTimeSleep as as *numeric const*$^{\text{(req. 7i)}}$

The parameters "analogueModels" and "decider" holds which analogueModels and decider to be used together with their parameters in XML formate. The exact formate still has to be declared!

4 Appendix