MiXiM - Physical Layer

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

21. 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:

- current signal strength or SNR information on channel
- current mode (RX, TX, SLEEP)
- current channel
- other?

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

- transmission over (send)
- other?

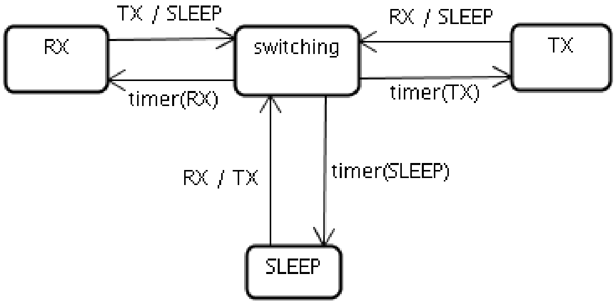

Figure 1: State machine for current mode

2.3 switch states

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

- current mode (RX, TX, SLEEP)
- current channel (1, 2, 3, ...)
- other?

2.4 send packets

The physical layer has to be able to send packets from the MAC Layer to the channel.

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 (maybe 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 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. packet is send to channel by physical layer

The following information is needed by the receiving physical layer:

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

2.5 receive packets

Following things have to be done with every received packet:

1. get transmission time of the packet and schedule transmission over event
2. apply *analog model* (simulate pathloss, shadowing, fading) to calculate signal strength over time

Every packet has to be classified as *signal*[1] or *noise*. The decision depends on one or more of the following points.

- signal strength (at signal start)

[1] We have some name collision here. With signal we mean every signal we receive over the channel. With *signal* we mean the signal we try to receive as a packet.

- are we already listening to a *signal* (most time we won't be able to listen to more than one *signal* at the same time, so every thing else is *noise*)

If the transmission of a *signal* is over we have to decide if it was received correctly. This is done with a *demodulation module* by evaluating the *signal to noise ratio* short *SNR*. If the signal was received correctly pass it (together with its SNR information) to the MAC Layer.

2.5.1 the analog model

The *analog model* has to be able to simulate the following things:

- pathloss
- shadowing
- fading

Further we set the following requirements to the *analog model*:

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

2.5.2 the demodulation module

We set the following requirements to the *demodulation module*:

- you should be able to set the *demodulation module* independent from physical layer
- you should be able to add your own *demodulation module*
- the *analog model* 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
- other?

2.7 parameters

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

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

3 modelling

4 Appendix

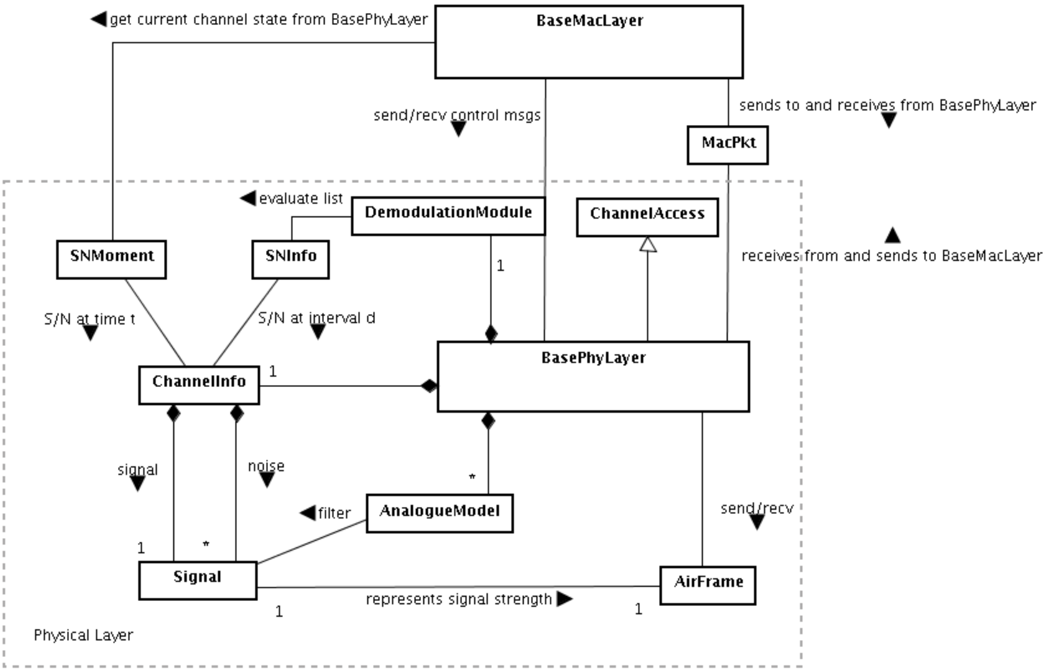

Figure 2: class graph

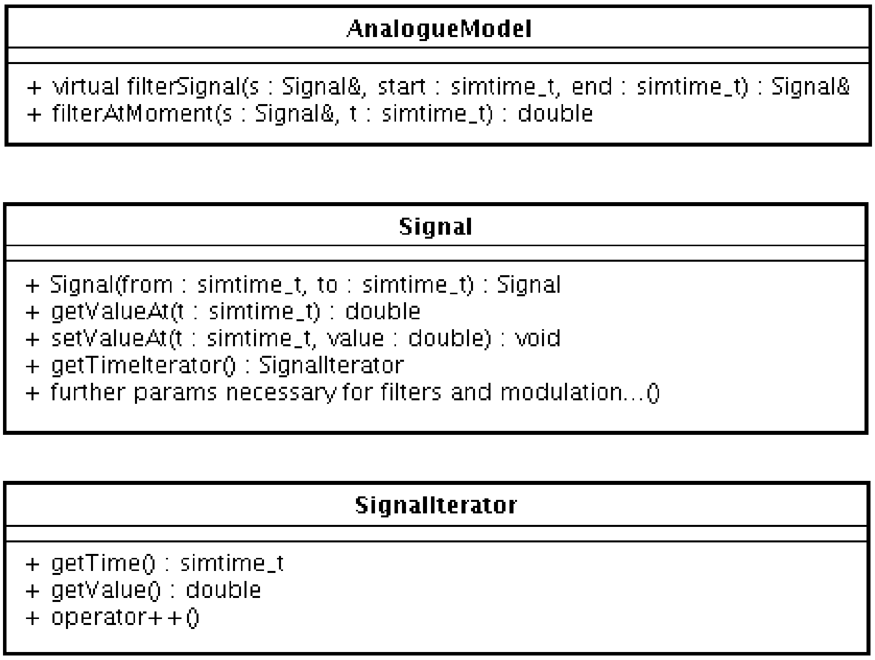
AnalogueModel
+ virtual filterSignal(s : Signal&, start : simtime_t, end : simtime_t) : Signal&
+ filterAtMoment(s : Signal&, t : simtime_t) : double
Signal
+ Signal(from : simtime_t, to : simtime_t) : Signal
+ getValueAt(t : simtime_t) : double
+ setValueAt(t : simtime_t, value : double) : void
+ getTimeIterator() : SignalIterator
+ further params necessary for filters and modulation...()
SignalIterator
+ getTime() : simtime_t
+ getValue() : double
+ operator++()

Figure 3: analogue model interface

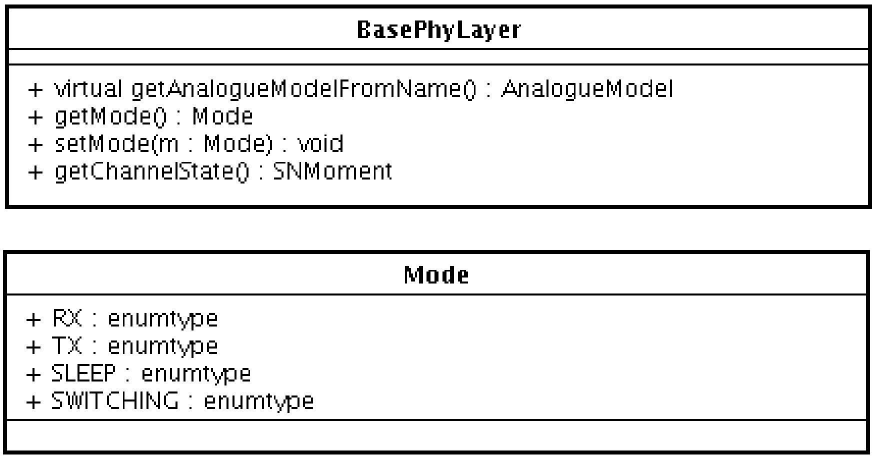
BasePhyLayer
+ virtual getAnalogueModelFromName() : AnalogueModel
+ getMode() : Mode
+ setMode(m : Mode) : void
+ getChannelState() : SNMoment
Mode
+ RX : enumtype
+ TX : enumtype
+ SLEEP : enumtype
+ SWITCHING : enumtype

Figure 4: BasePhyLayer interface

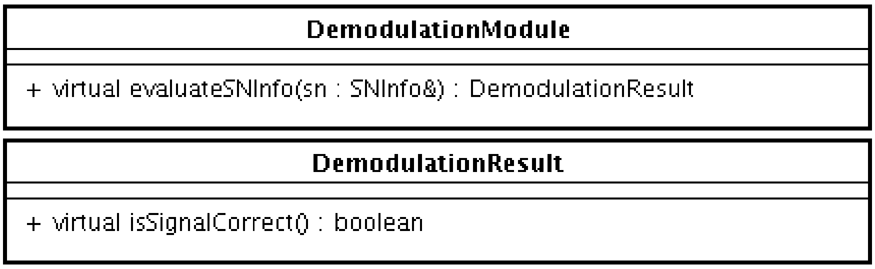
DemodulationModule
+ virtual evaluateSNInfo(sn : SNInfo&) : DemodulationResult
DemodulationResult
+ virtual isSignalCorrect() : boolean

Figure 5: Demodulator interface

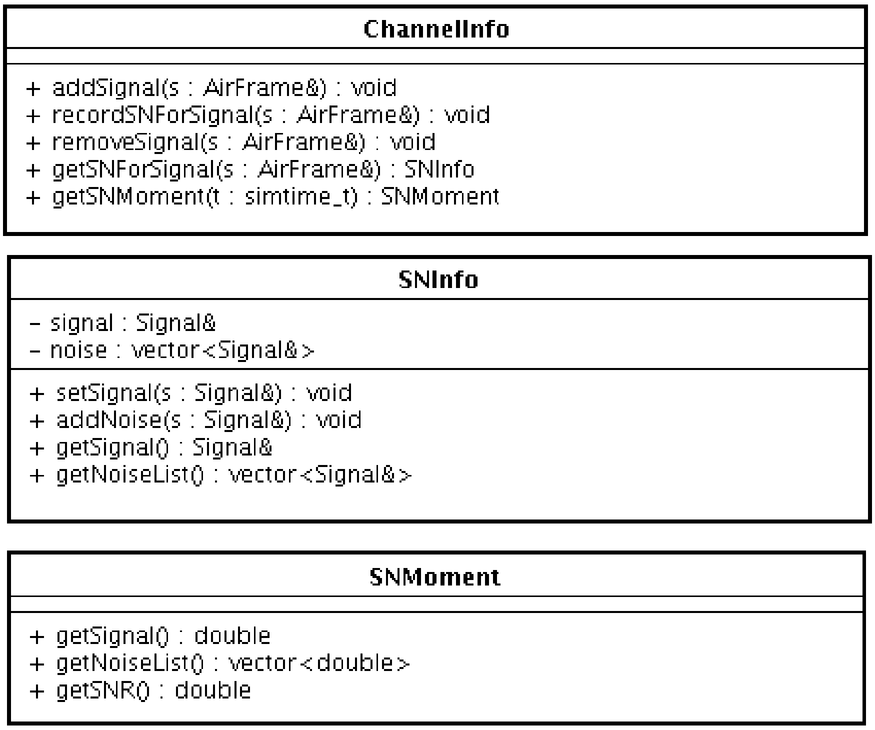
ChannelInfo
+ addSignal(s : AirFrame&) : void
+ recordSNForSignal(s : AirFrame&) : void
+ removeSignal(s : AirFrame&) : void
+ getSNForSignal(s : AirFrame&) : SNInfo
+ getSNMoment(t : simtime_t) : SNMoment
SNInfo
– signal : Signal&
– noise : vector<Signal&>
+ setSignal(s : Signal&) : void
+ addNoise(s : Signal&) : void
+ getSignal() : Signal&
+ getNoiseList() : vector<Signal&>
SNMoment
+ getSignal() : double
+ getNoiseList() : vector<double>
+ getSNR() : double

Figure 6: channel details

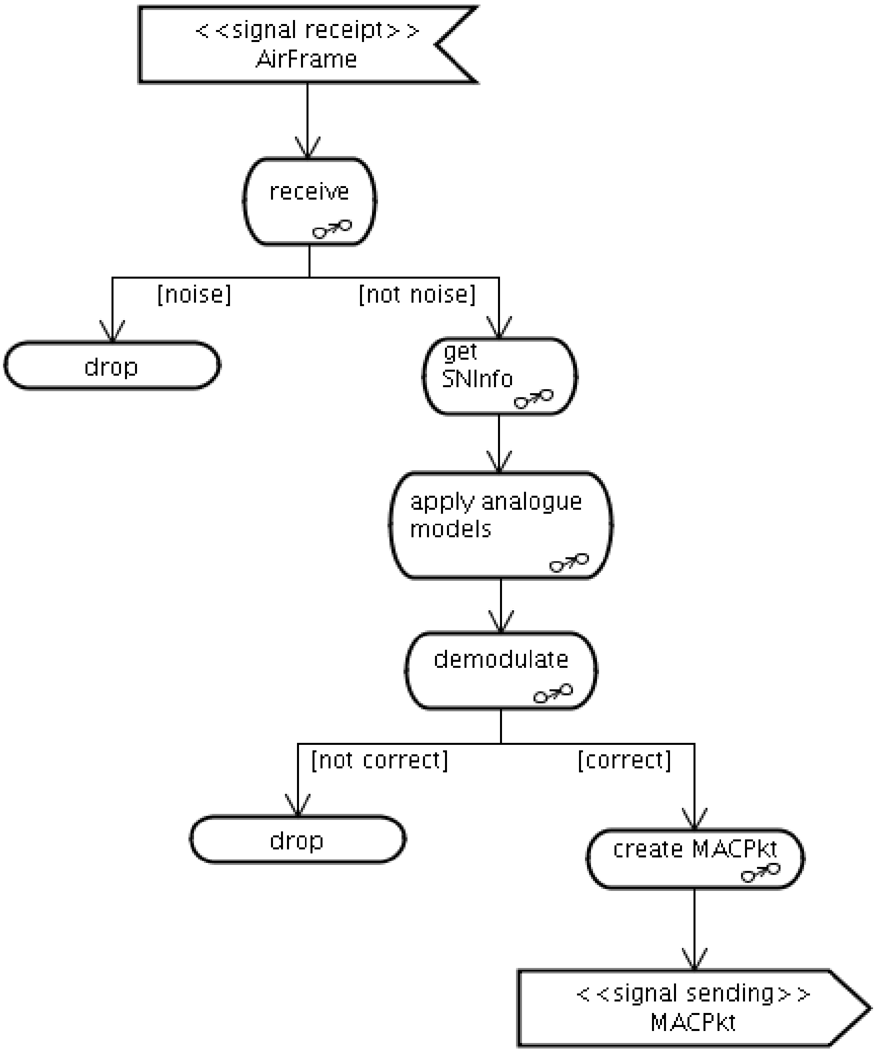

Figure 7: receiving process

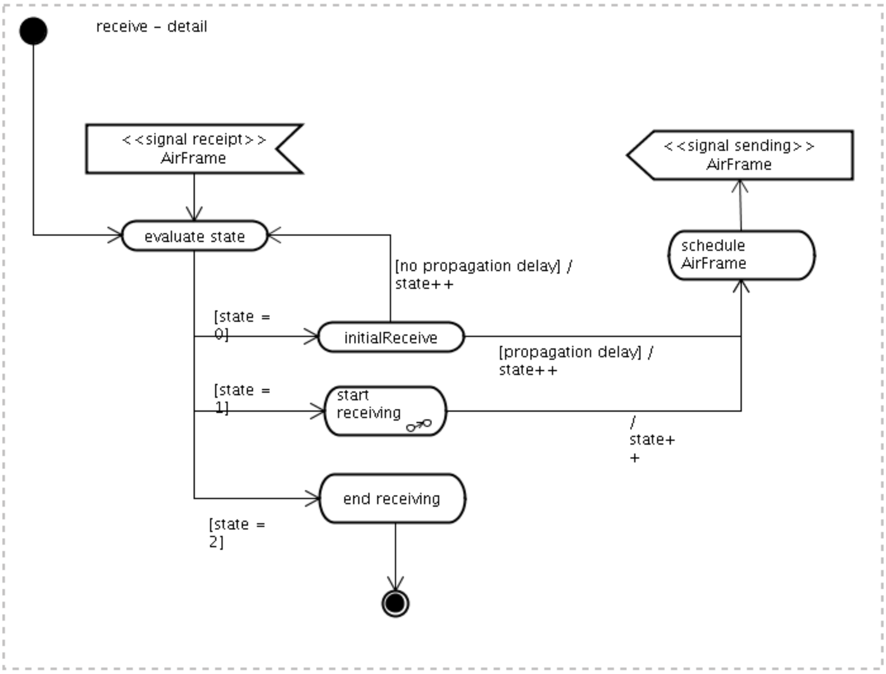

Figure 8: receive detail

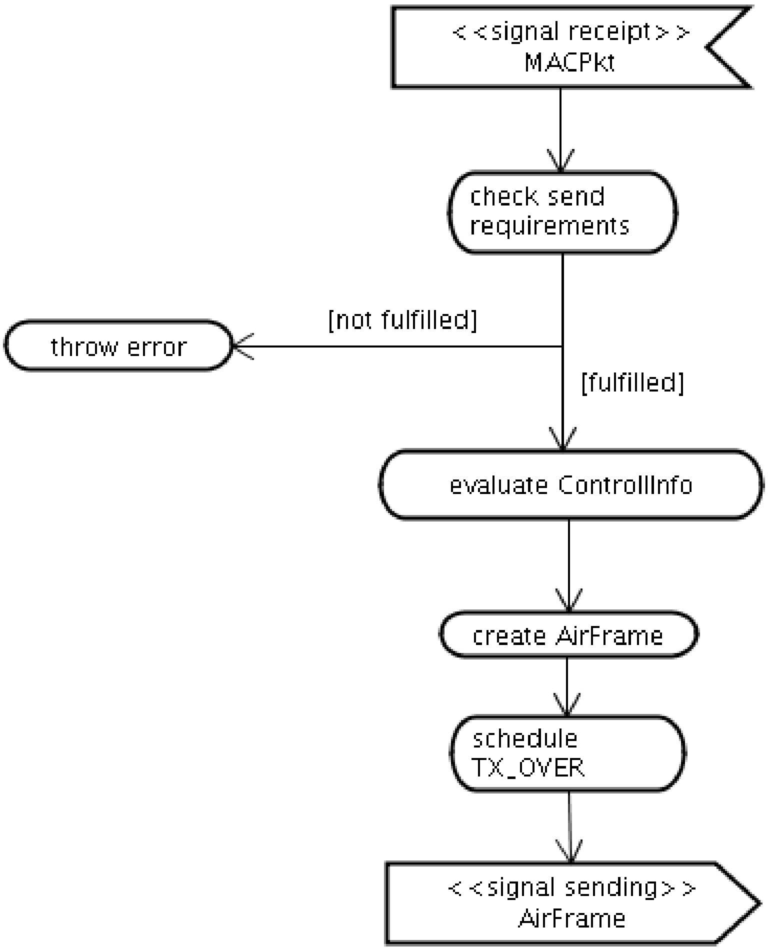

Figure 9: sending process