MiXiM - Physical Layer

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

31. 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)

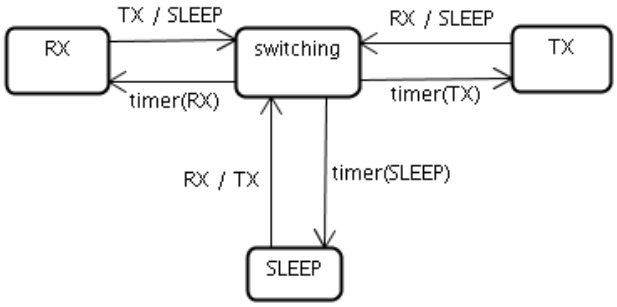

Figure 1: State machine for current mode

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 **(see 2.6)**strength

   (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

3.1 overview

Here we present the design- and interface details of the OMNeT-module **BasePhyLayer** and go step-by-step through the requirement specification:

1. internal class diagram of **BasePhyLayer** and relation (pointers, references and OMNeT-gates) 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. detailed flow chart for the receiving process

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 OMNeT-messages sent between them.

The **BasePhyLayer** hold a list$^{(\text{req. } 4(\text{h})\text{v})}$ 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$^{(\text{req. } 4(\text{h})\text{vii})(\text{req. } 5\text{d})}$ and replace$^{(\text{req. } 4(\text{h})\text{vi})(\text{req. } 5\text{c})}$ them independently from the **BasePhyLayer**.

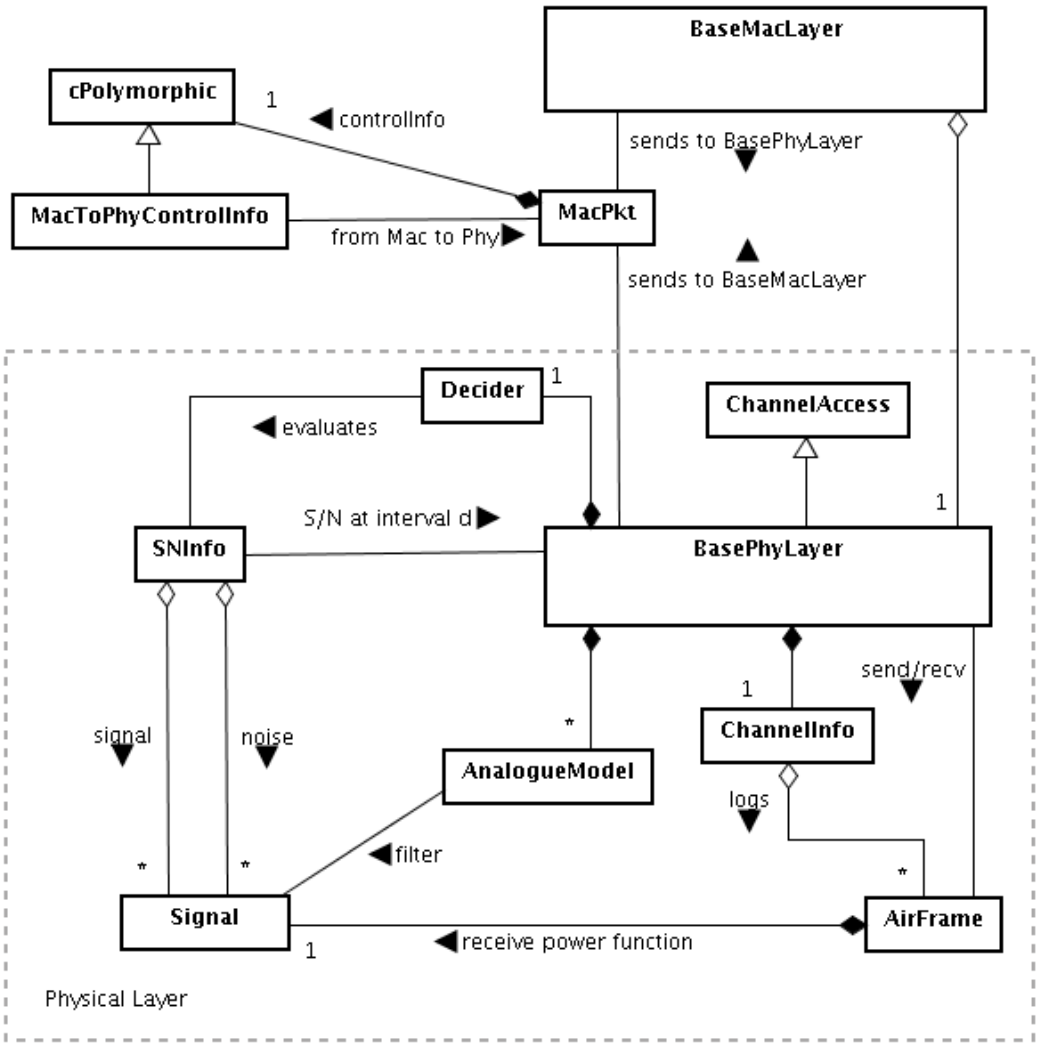

Figure 2: class graph

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**. The **BaseMacLayer** is connected by a control channel to the **BasePhyLayer**. Over this channel the **BasePhyLayer** can inform the **BaseMacLayer** about specific events(req. 1b) (like the transmission over event).

Additionaly **BaseMacLayer** holds a reference to the **BasePhyLayer**. With this reference **BaseMacLayer** can obtain information(req. 1a) about the channelstate(req. 1c) and the current mode(req. 1d). Further it is able to set current mode in **BasePhyLayer** via method call(req. 2a).

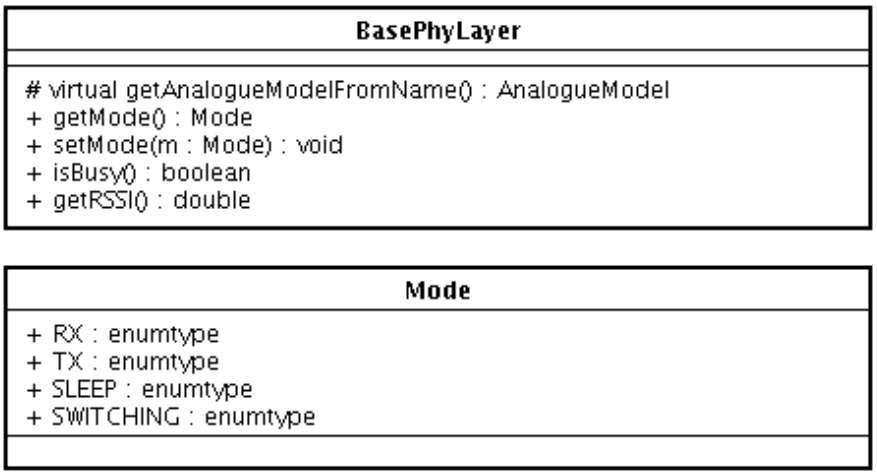

Figure 3: BasePhyLayer interface

3.4 AnalogueModel and Signal

The Signal represents the function of signal strength over time of an AirFrame. The AnalogueModel is designed as a filter which can be applied at a Signal[req. 4(h)i]. This way attenuation effects like pathloss[req. 4(h)ii], shadowing[req. 4(h)iii] and fading[req. 4(h)iv] can be implemented as a concrete AnalogueModel which changes a given Signal appropriatly.

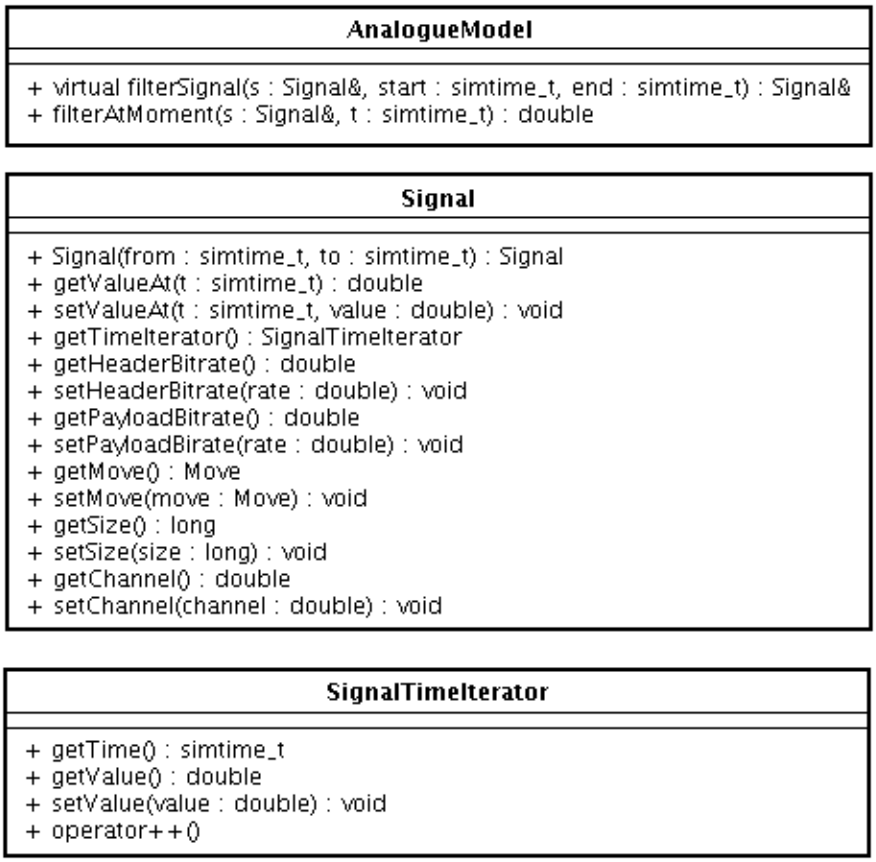

Figure 4: analogue model interface

To be able to filter a referenced signal in a specified interval or at a specific point in time, the AnalogueModel gets an iterator from the Signal.

> *BEWARE: Anyone who subclasses Signal should make shure to have a properly working SignalTimeIterator (subclassed) for it. The SignalTimeIterator should always iterate over every timestamp in each dimension. This way simple filters can be reused for multidimensional Signals.*

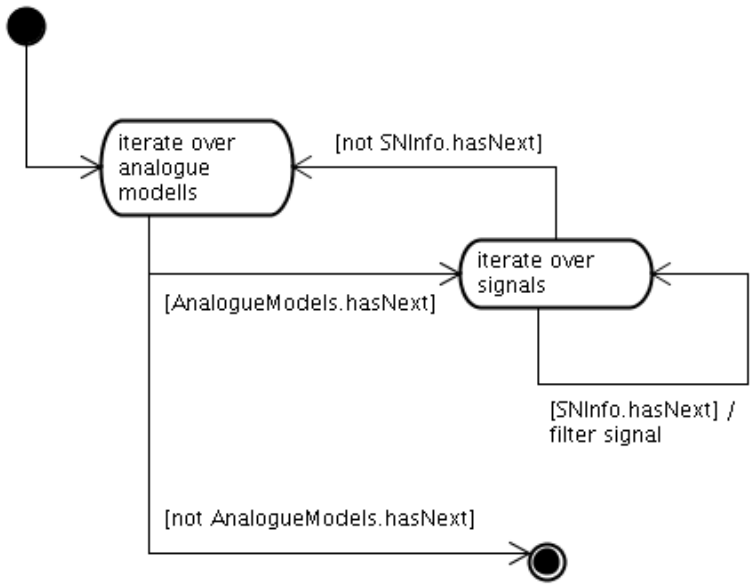

Figure 5: application of analogue models

3.5 Decider

The decider provides one method to decide wether a packet could be interpreted as Signal or it is only noise(req. 5a). And another method to decide if a packet was received correctly(req. 5b).

The deciders performs both tasks by evaluating the Signals and noise during a certain interval. This Signal and noise information is hold inside the SNInfo class (see 3.6).

To be able to provide additional information (like bit wise errors(req. 5e)) besides the information if the packet is correct, the Decider returns a Decider-Result.

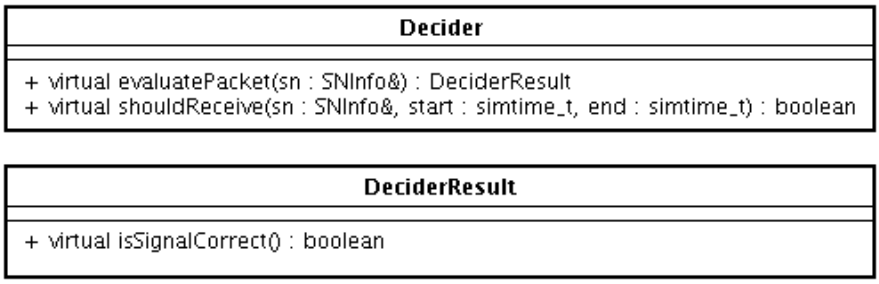

Figure 6: Decider interface

3.6 SNInfo and ChannelInfo

ChannelInfo keeps track of all AirFrames on the channel. It does not differ between *signal* and *noise*. **BasePhyLayer** is able to add and remove references to certain AirFrames.

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.

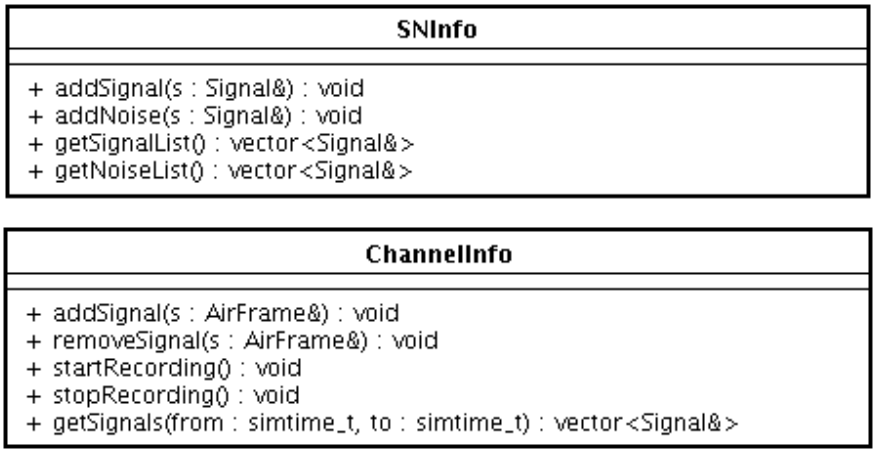

Figure 7: channel details

3.7 AirFrame

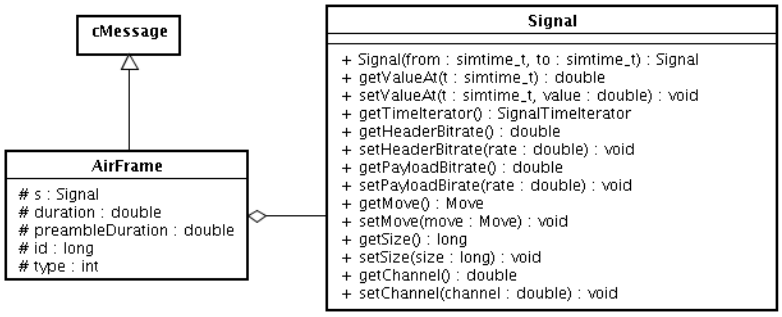

Figure 8: member arrangement in AirFrame and Signal

3.8 receiving a MacPkt

The class MacToPhyControlInfo is designed as the container for control info(req. 3a) the **BaseMacLayer** wants to attach to the packet given down to **BasePhyLayer** for sending.

The packet itself is handed down as a MacPkt via OMNeT-channel.

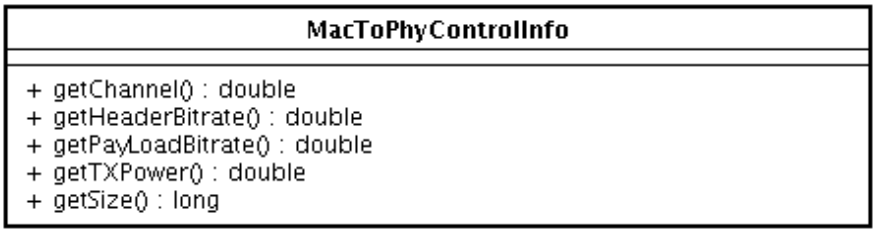

Figure 9: MacToPhyControlInfo interface

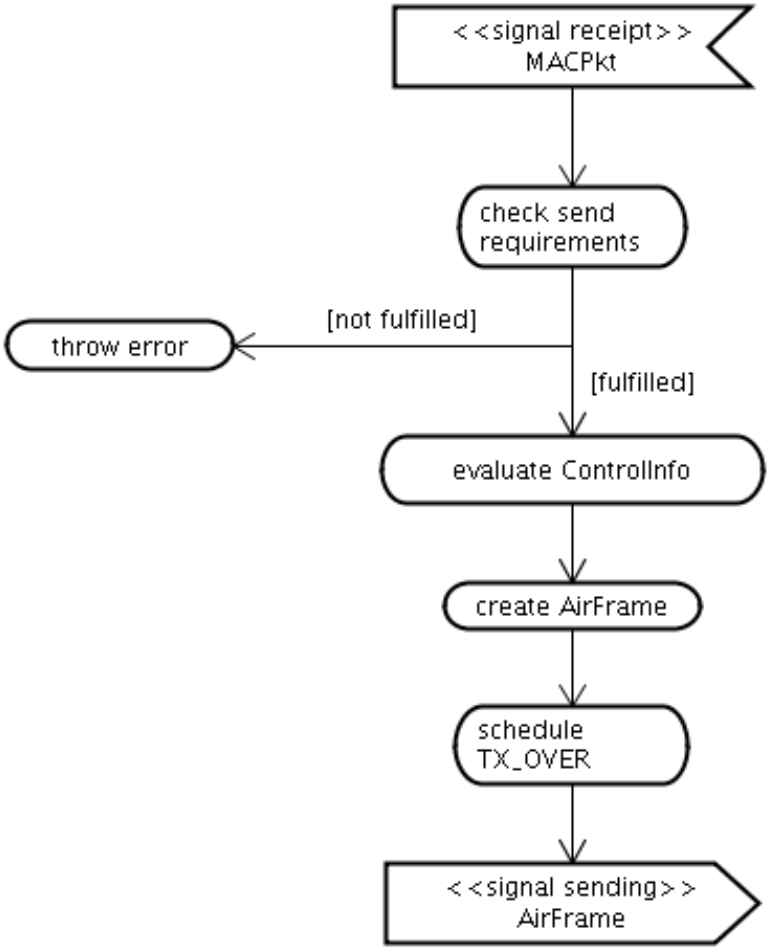

Figure 10: sending process

3.9 Receiving and processing an AirFrame

The reception of an AirFrame is divided into:

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

Afterwards the packet is dropped if it was noise. Otherwise Signal and all interfering noise is collected within a SNInfo, filtered by the analogue models(req. 4f) and then proceeded to the Decider to decide if it was received correctly. If the result turns out positive, the MacPkt inside the AirFrame is decapsulated and send to the **BaseMacLayer**(req. 4g).

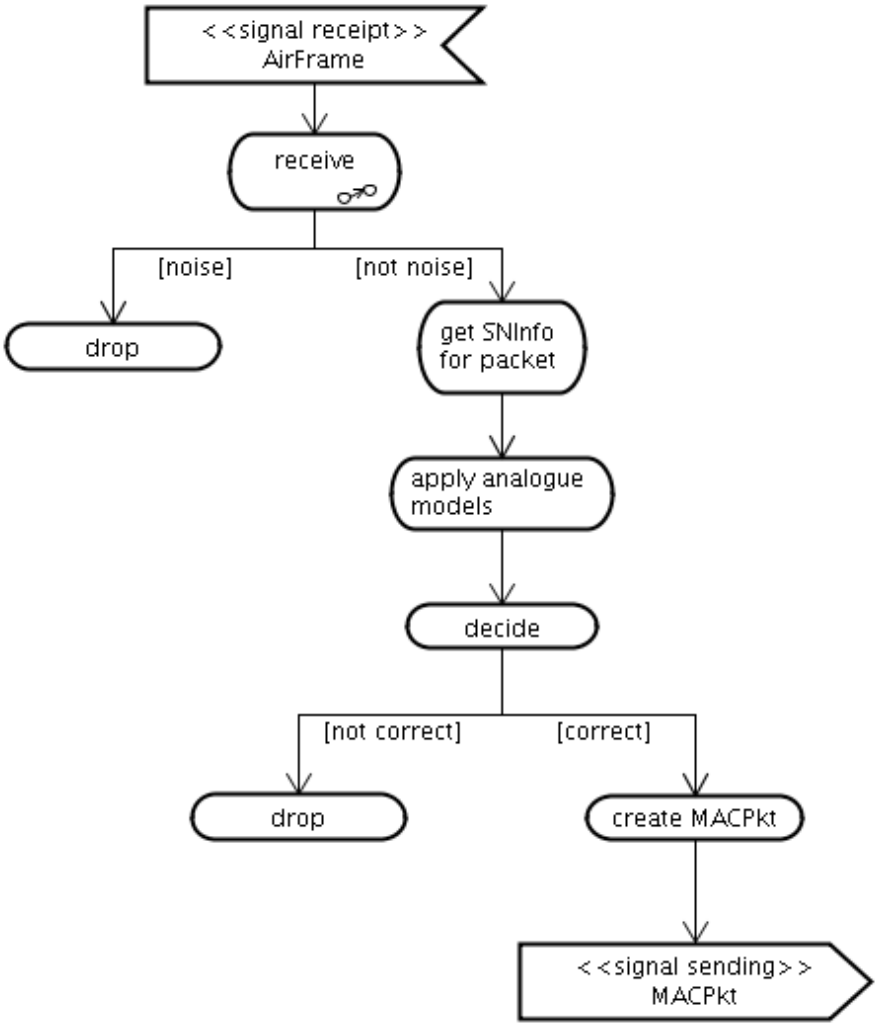

Figure 11: receiving process

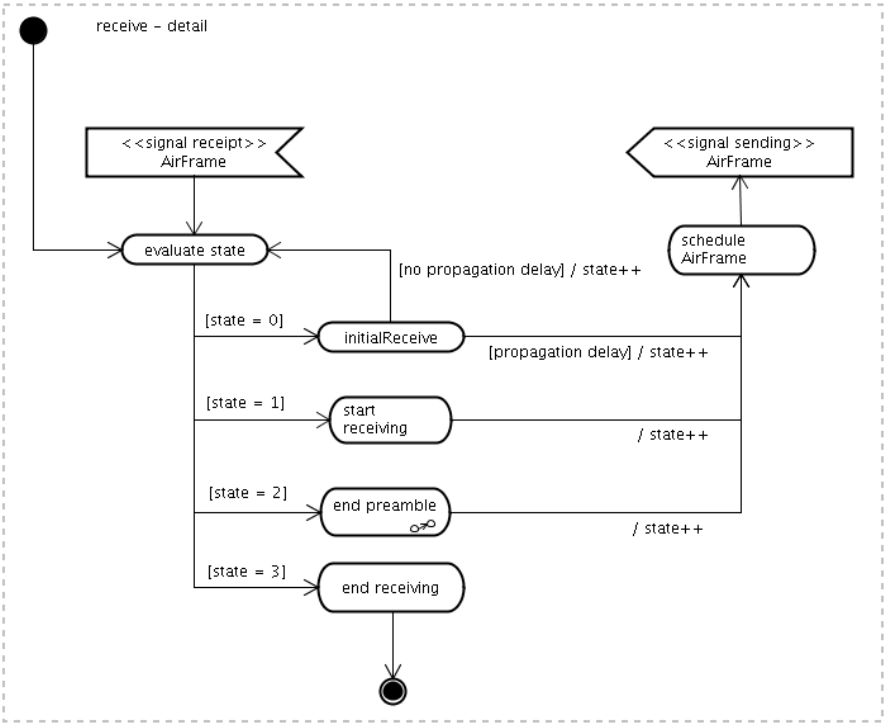

Figure 12: receive detail

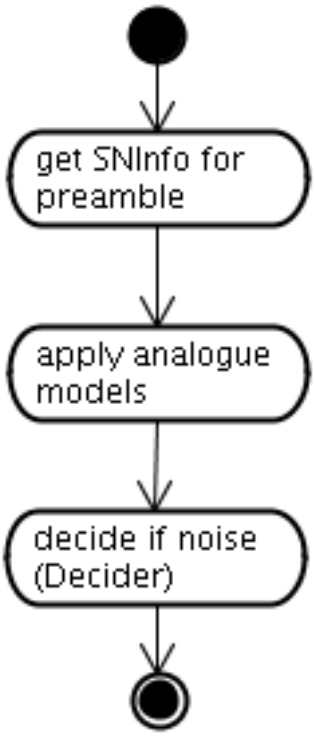

Figure 13: end preamble detail

4 Appendix