MiXiM – The Physical Layer
An Architecture Overview

Karl Wessel, Michael Swigulski, Andreas Köpke, Daniel Willkomm
Technische Universität Berlin, Telecommunication Networks Group
Berlin, Germany
{wessel, swigulski, koepke, willkomm}@tkn.tu-berlin.de

ABSTRACT
Simulating the physical layer of wireless communication remains a challenge. Communication standards like OFDM or MIMO systems go beyond the simple single narrow frequency band, single antenna model used in popular simulators. Yet, these technologies gain popularity, since they provide researchers with a plethora of possibilities that can be explored to invent new protocols or improve existing ones. However, building a detailed and sufficiently accurate model for such complex systems is a tremendous task that takes a lot of time. In this paper we present the physical layer model of MiXiM, which tackles this task. It provides the researcher with an easy to use interface to the wireless transmission medium. It models the wireless medium in all three dimensions (time, space and frequency) supporting the implementation of future wireless communication standards, but at the same time also supports easy modeling and simulation of traditional single frequency systems.

1. INTRODUCTION
MiXiM [1] has been introduced [3] as a very powerful extension to simulate wireless and mobile networks using the discrete event simulator OMNeT++ [4]. MiXiM aims to provide the developer with a powerful and feature-rich toolbox to enable and facilitate the simulation and performance analysis of wireless networks. At the same time the structure and design of MiXiM is such, that it tries to hide the complexity of such simulations and provides the developer with a clean and easy to use interface. This approach facilitates the development and implementation of specific models and protocols for wireless communication without having to worry about the underlying architecture more than necessary.

In order to gain a deeper understanding of the complexity of the problem, let us start with the main components that are responsible for the transmission process, shown in Figure 1. In this figure, we consider the transmission of a single packet, concentrating on the interaction of the components. Assume, the sending Medium Access Control (MAC) protocol received a packet that shall be transmitted. After packing this into a Service Data Unit (SDU), the MAC protocol hands this packet down to the physical layer for transmission, together with some information on how the packet

shall be transmitted. The physical layer uses this information to compute a signal that represents the packet. This signal is transmitted via the wireless medium, where it gets distorted by multiple influences, starting from attenuation due to various causes as well as other interfering transmitters. The physical layer of the receiver receives this distorted signal and has to derive a binary representation that hopefully resembles the original SDU. This is passed to the receiving MAC layer, together with some meta information like the received signal strength indicator (RSSI) of the packet.

In this paper we present the physical layer of MiXiM. While many popular simulators only model single frequency, single antenna systems, such models are not sufficient anymore. The physical layer of MiXiM is designed with flexibility in mind without sacrificing efficiency. It can be used for simple single frequency, single bit rate systems used for instance in sensor network simulations, multiple channels, multiple bit rate systems like IEEE 802.11b that sends the header and the payload of the packet with different bit rates, systems that change the transmission frequency between each packet like Bluetooth, Orthogonal Frequency Division Multiplexing (OFDM) systems like 802.11a that transmit in parallel on multiple frequencies and Multiple Input Multiple Output (MIMO) systems that use multiple antennas for the transmission and the reception.

In addition to this wide range of different transmission standards, there are many different models for the wireless channel, each concentrating on a different effect in a certain environment. There are path loss models that attenuate the transmitted signals according to the traveled distance, abstract models for shadowing effects due to obstacles like the log-normal fading, models for fast fading due to the mobility of the nodes like Rice and Rayleigh fading and many more.

To complicate things further, standards like 802.11g include a Forward Error Correction (FEC) and the design of the physical

![Figure 1: Aspects of a wireless transmission](image-url)
layer of MiXiM should not prevent research on the influence of such codes. This means that all the attenuation effects should be computable on a sub-packet time scale.

In this paper, we present how the physical layer supports all these aspects of a wireless transmission. Section 2 gives an overview of the physical layer structure in MiXiM and explains how the various aspects of a wireless transmission. Section 2.1 BasePhyLayer

The BasePhyLayer provides the general structure for a physical layer module in MiXiM. The AnalogueModels and Decider modules to be used by a BasePhyLayer are defined in a config.xml file which contains the names of them as well as the needed parameters. For AnalogueModel and Decider implementations already included in MiXiM this is sufficient, own implementations have to be introduced by sub-classing BasePhyLayer and proper initialization in the overloaded getDeciderFromName() - or getAnalogueModelFromName() -method. A new Radio module can be plugged into the physical layer by sub-classing BasePhyLayer and overloading the initializeRadio()-method to properly initialize the new Radio module.

One of the main tasks of the BasePhyLayer is the processing of messages (AirFrames). Whereas the sending of messages is straightforward and only requires the BasePhyLayer to set some parameters and calculate the propagation delay (if necessary), the receiving process is more complicated. Figure 3 shows a state diagram of the receive process. Note, that sending an AirFrame in MiXiM means that it is received by all nodes which potentially are in the interference range of the sending node. It is the nodes’ task to figure out which AirFrames are (a) strong enough to be received and (b) are actually destined for the node. For details, refer to [3].

When the reception of the AirFrame starts (i.e. the first bit arrives), it is first of all registered with ChannelInfo, which keeps track of all AirFrames on the channel (see Section 2.2 for details). After that all AnalogueModels are applied to the Signal contained in the AirFrame (see Section 3). The AirFrame is then handed over to the Decider, which reschedules the AirFrame to the time it will make a decision whether to receive and further process the AirFrame or to treat it as noise. For details on the decision process refer to Section 4.1.
2.2 ChannelInfo

The ChannelInfo module is used to keep track of the AirFrames on the channel. As soon as an AirFrame starts (i.e. the first bit arrives), it is added to ChannelInfo and considered active until it ends. Once ended, it becomes inactive for ChannelInfo but is still kept as interfering AirFrame until there is no more active AirFrame intersecting with it in time. The AirFrames have to be kept in order to be able to calculate the signal to interference plus noise ratio (SINR) of a specific AirFrame currently being received (for which all other AirFrames on the channel are interference). ChannelInfo is able to return all AirFrames currently on the channel intersecting with a given, not outdated, time-interval. This service is used by the Decider to calculate SINR values.

![Figure 3: Receiving an AirFrame](image)

Figure 3: Receiving an AirFrame

Consider the simple scenario in Figure 4 where the AirFrames A, B and C arrive one after another. They are handled by ChannelInfo as follows:

\[ t_0 \]: A starts and becomes active.

\[ t_1 \]: B starts and becomes active.

\[ t_2 \]: A ends and becomes inactive but is not yet deleted since it intersects with B, which is still active.

\[ t_3 \]: C starts and becomes active.

\[ t_4 \]: B ends and becomes inactive but is not yet deleted since it intersects with C, which is still active; A is deleted since there is no (more) active AirFrame it intersects with.

\[ t_5 \]: C ends and is deleted since there is no active AirFrame it intersects with; B is deleted since C is not active anymore.

Given the above example, consider B to be an AirFrame which is actually being received (not treated as noise) and thus evaluated by the Decider (at time \( t_5 \)). Since AirFrame B as well as the interfering AirFrames A and C have to be taken into account in order to calculate the SINR, all of those have to be accessible to the Decider. This is why ChannelInfo does not delete an AirFrame as long as there is an active one overlapping in time (in the example, A is not deleted as long as B is active). Thus ChannelInfo is able to hand over a currently received AirFrame (B) together with all interfering AirFrames (A and C) on demand.

2.3 Signal Concept

Every AirFrame contains a Signal which describes the actual physical signal sent over the channel. While it is created at the senders MAC layer, it is heavily affected by the receivers AnalogueModels which model the influences of the wireless environment it travels through.

To represent a physical signal the Signal class contains Mappings for transmission power, bit-rate, attenuations and receiving power which all represent mathematical mappings defining the according data over time and maybe more dimensions like frequency or space. The lower part of Figure 2 shows the relationships of the different classes: Each AirFrame contains a Signal object, which in turn contains various Mapping objects describing the different properties of the physical signal.

While transmission power and bit-rate Mappings are added to the Signal by the MAC layer module of the sender, the attenuation Mappings are defined and added by the AnalogueModels used at the receiving physical layer module. The receiving power Mapping is defined by multiplying all of the attenuation Mappings element-wise with the transmission power Mapping. To avoid unnecessary calculations this is done only on demand. The receive power is used by the Decider to calculate SINR and to perform the bit-error analysis. Details of the Mapping class and its implementation can be found in Section 5.

3. RADIO AND ANALOGUE MODELS

In this section we describe how the influence on a Signal on its way from the sender to the receiver(s) is considered and realized in our model. When transmitted by the wireless channel, a Signal is attenuated due to several effects which are modeled and applied to the Signal by AnalogueModels. Furthermore the state of the receivers Radio possibly affects the transmitted Signal. So this state is represented by the RadioStateAnalogueModel and applied accordingly.

3.1 Radio

The Radio of a NIC is part of the physical layer-module and implemented as a state-machine. There are two main functionalities for the Radio module: simulation of radio switching times and simulation of the effects a radio has on signal reception.

In state-of-the-art hardware, the time it takes to switch a radio from transmit to receive or sleep are usually very small and thus are often neglected in simulation. However, there are scenarios where switching times should not be neglected, since they can have significant influence on the simulation results. In MiXiM we thus support both scenarios: simulation of switching-times for a radio-state transition as well as zero-time switching. The MAC layer can switch the Radio to another state by calling setRadioState() on the MacTo PhyInterface. In case of a non-zero switching times it is
(b) radio state analogue model (RSAM) mapping

(d) resulting RX-power mapping

Figure 5: TX-power mapping multiplicated with RSAM mapping

notified by a control-message of kind \texttt{RADIO\_SWITCHING\_OVER} as soon as the simulated switching process has finished. For zero switching times, there are no control messages sent and thus the performance (speed) of the simulation is not degraded.

Most radios are half-duplex, meaning they can only send or receive at a time, but not concurrently. However, there also exist duplex radios which can concurrently send and receive. The standard radio module in MiXiM is half-duplex but can be easily extended for full-duplex operation. In the following, we will focus on half-duplex radios.

The state of the radio affects a received signal. If in send, sleep or off mode, no signal can be received. In MiXiM, this is represented by the \texttt{RadioStateAnalogueModel} which returns dynamically created attenuation-step-mappings. The \texttt{Radio} tracks the states it is in over time and maps the several states to attenuation values. If the \texttt{Radio} is in the receive state, there is no attenuation, otherwise the signal is attenuated by 100\%. This concept can be easily extended to support e.g. two independent antennas by exploiting the spatial dimension of the attenuation mapping.

3.2 Analogue Models

The \texttt{AnalogueModel}s are used to simulate effects like path-loss, shadowing and fading which affect the signal during its transmission. A physical layer can contain an arbitrary number of \texttt{AnalogueModel}s, which can be plugged-in as described in Section 2.1. A \texttt{Signal} arriving at the physical layer will be subsequently processed by all \texttt{AnalogueModel}s.

The \texttt{AnalogueModel} class itself is only an interface defining a \texttt{filterSignal()} method which takes the \texttt{Signal} to be processed as an argument. Actual implementations of the interface implement this method by taking the information they need from the passed \texttt{Signal} (like start and end time of the \texttt{Signal} or the \texttt{HostMove} of the sender), create an attenuation Mapping which describes the attenuation caused by the effect this \texttt{AnalogueModel} simulates and add it to the \texttt{Signal}'s attenuation list. To describe the attenuation mapping for the signal the \texttt{AnalogueModel} can use an own Mapping implementation or only calculate the attenuation values of some points of interest and put them into one of the default Mapping implementations. For details refer to Section 5.

Figure 5 shows an example of a TX-power signal, which is filtered by two \texttt{AnalogueModel}s. The step-function in Figure 5(b) represents the attenuation of a \texttt{RadioStateAnalogueModel}. The state of the radio changes at $t = 0.2$ to RX and at $t = 3.05$ back to TX or SLEEP. In Figure 5(c) a simple path-loss \texttt{AnalogueModel} is shown. The resulting RX-power is shown in Figure 5(d).

Note, that we represent attenuation as a quantity between 0 and 1 without a unit. An attenuation of 100 \% is represented as 0 and no attenuation (0 \%) as 1. The advantage of this approach is that the resulting RX-mapping can be simply calculated by multiplying the TX-mapping with all attenuation mappings.

4. DECIDER

The decider is the evaluation component of the physical layer. It is responsible for evaluating received \texttt{Signals}, which includes categorizing arriving \texttt{Signals} as interference or messages to receive and, at the end of receiving a message, calculating the bit-errors for that message. Another task of the decider is the evaluation of the channel, i.e. to perform carrier sensing and report the channel state to the MAC layer.

4.1 AirFrame Evaluation

Each \texttt{AirFrame} is processed several times by the \texttt{processSignal()} method of the decider, as indicated in Figure 3. The number of times it is processed (and also how it is processed) actually depend on the specific \texttt{Decider} implementation. However, the example below represents the most common case and also reflects the mandatory processing events for each \texttt{AirFrame}.

1. Upon the reception of the first bit of an \texttt{AirFrame} the decider has to process the \texttt{Signal} of that \texttt{AirFrame} for the first time. It basically only has to decide, at which point in time it can decide whether the packet should be treated as noise or not and returns this time-point to the physical layer. Usually, the time to decide whether to receive an \texttt{AirFrame} or treat it as interference / noise is after the physical layer preamble is completely received. However, the \texttt{Decider} can also make the decision immediately or at any other point in time.

2. The next time the decider gets the \texttt{Signal}, it has to decide whether the \texttt{AirFrame} is considered interference or whether it is actually being received. This is done by evaluating the
sensing the channel for the specified amount of time and answer by a signal is received at the moment of the request.

4.2 Channel Sensing

As mentioned above the decider is also responsible for evaluating the wireless channel, i.e. deciding whether the channel is busy or idle at a specific point in time. This is an important functionality provided to the MAC layer which is needed for Carrier Sense Multiple Access (CSMA) protocols.

In a real system such a channel sensing takes time, however this time is often negligible an thus not accounted for in simulation. In the example of Figure 4 this means that also the receive power of AirFrame C has to be calculated.

How the SINR is used to determine the bit-errors of the AirFrame depends on the specific Decider implementation. In the simplest case, the SINR is just checked against some threshold and the AirFrame is considered lost if the threshold is exceeded. Another option is to determine a bit-error probability based on bit-error curves.

The Decider has to pass the decapsulated MAC-packet up to the MAC layer together with a DeciderResult. The DeciderResult contains the evaluation data for the MAC-packet (packet lost / correctly received, number of bit-errors, etc.) where the specific evaluation data again depends on the concrete Decider implementation.

5. MAPPING

Many aspects of simulating the wireless transmission process can be expressed by mathematical mappings and operations on them. The signal sent to the channel might consist of a mapping for the power $f_{tx} : \text{time} \times \text{frequency} \times \text{space} \rightarrow \text{power}$ and one for the bit-rate $f_{rate} : \text{time} \times \text{frequency} \times \text{space} \rightarrow \text{bitrate}$ (in MiXiM the bit-rate represents the coding / modulation combination used for a signal). The attenuation defined by the AnalogueModels can be represented by further mappings $f_{att} : \text{time} \times \text{frequency} \times \text{space} \rightarrow \text{attenuation}$ defining an attenuation factor varying in time, frequency and space. The resulting receiving power can then be calculated by multiplying the attenuation mappings element-wise with the power mapping which results in another mapping $f_{rx} : \text{time} \times \text{frequency} \times \text{space} \rightarrow \text{power}$ where $f_{rx} = f_{tx} \times \prod_{f_{att}}$. Further the SINR needed by most Deciders to evaluate if a signal was received correctly is calculated by dividing the receiving power of the signal to evaluate ($f_{rx}$) by the receiving power of the interference, which is the summed up receiving power of every signal interfering with the signal to be evaluated. So we get further element-wise operations on mappings: $f_{slnr} = \frac{f_{rx}}{\sum f_{rx}}$. The Decider then has to evaluate the different SINR values in time, frequency and space inside the SINR mapping.

The mapping abstraction enables us to compute the SINR of a received packet at any interesting point in time, frequency and space in a consistent fashion. For systems that are not as complex, the consistency of the abstractions allows us to use much simpler models without paying an inadequate price for the generalization.

In the following sections we describe how we implement the mapping abstraction and the necessary operations in MiXiM.

5.1 Requirements for a Mapping

In order to define a Mappings class for MiXiM, we first need a list of requirements it has to fulfill.

First of all, the Mapping class should support almost arbitrary domains. The simplest case is just a mapping from time to power or bit-rate, i.e. $f : \mathbb{R} \rightarrow \mathbb{R}$. But we can also have mappings defined in time, frequency and space $f : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$. While the codomain stays the same the domain can change. Generally, we can assume that a codomain of a single real value is sufficient because if we need to represent something like $f : \mathbb{R} \rightarrow \mathbb{R} \times \mathbb{R}$ we could just use two separate mappings $f_1 : \mathbb{R} \rightarrow \mathbb{R}$ and $f_2 : \mathbb{R} \rightarrow \mathbb{R}$ to achieve the same.

Naturally, the Mapping class also has to provide the mapping functionality itself, meaning given a number of arbitrary argument values in the mappings domain it has to return the specific mapped value for these arguments. This can either be done by using the passed arguments and maybe some other parameters to evaluate a mathematical formula and return the result, or by a lookup of the value for the specified argument. Note, that if there is no value defined for this specific argument, the Mapping class also has to be able return a value by interpolating between adjacent arguments.

At last, since the Decider has to evaluate the values of an SINR mapping but probably can’t check every possible point in time, frequency and space, the Mapping class has to provide the possibility to iterate over a set of given arguments.

5.2 Mapping Class Overview

The above requirements led to the modeling of the Mapping interface as shown in Figure 6. The most important class members are shown in Figure 7.

A single dimension (like time) is represented by the Dimension class. The domain of a Mapping can consist of more than one di-
dimension (e.g. time and frequency) and is defined by the DimensionSet class which contains one or more Dimension objects. This enables us to define arbitrary domains and fulfill the above requirements.\footnote{Note, that we usually refer to the domains “time”, “frequency”, and “space” in this paper because these will be the domains mainly used in MiXiM.} Externally, a Dimension is represented by a case sensitive name (like “frequency”), internally a simulation wide static unique id is used. Since this id is used to define the iteration order for multi dimensional mappings (see section 5.3) it is important to know that the time dimension always gets zero as id and every other id is an increasing integer assigned at the first creation of a Dimension object for a dimension.

To be able to define positions in Mappings with arbitrary domain we define the Argument class. It maps from every Dimension of a domain to a value defining the position of the Argument instance in that dimension.

The Mapping and ConstMapping classes are interfaces which define the methods a class has to provide to be able to represent a mathematical mapping. The ConstMappingIterator and MappingIterator classes are interfaces as well and define the methods provided by an iterator for a mapping to be able to iterate over the points of interest of a mapping. The use of interfaces separates the information of how the data of a mapping is created from the way it is accessed. This enables us to provide methods which are generally applicable for any mapping (e.g. mathematical operations). It also allows implementing different kinds of data sources for a mapping (e.g. a mathematical formula or a number of interpolated key entries).

The difference between the Const- and non-Const-version of the interfaces is that the Const-version misses the setValue()-method so implementations do not have to provide a way to change the represented mapping arbitrarily. This is useful if a formula is used to represent the mapping where the parameterization can be changed but not the values themselves directly.

An important design decision we made is that every Mapping has to contain the time in its domain. Since signals are always defined over time and almost any mapping refers to a signal this does not constrain the user too much while it enabled us to use the more accurate `sintime_t` as type for positions in the time dimension.

5.3 Mapping Iteration

While it is trivial to iterate over a one dimensional mapping because there exists an unique order of the values, iteration over multidimensional mappings is more challenging.

We used the ordering of the Dimensions at this point to get a well defined iteration order every MappingIterator has to provide. A mapping with a multidimensional domain, e.g., \( f: \text{time} \times \text{freq} \times \text{space} \rightarrow \mathbb{R} \) can also be represented by a nesting of mappings with a one dimensional domain \( f_1: \text{space} \rightarrow (\text{freq} \rightarrow (\text{time} \rightarrow \mathbb{R})) \). So \( f \) would map from a number of points in space to a number of “sub-mappings” which again map from a number of points in frequency to a number of further “sub-mappings” which finally map from a number of points in time to an actual value.

The ordering feature of the Dimension class is used to define the ordering used to iterate over the mapping. Let us assume the dimensions are ordered \( \text{space} > \text{frequency} > \text{time} \). We would start iterating at the first position in space and directly go down to the according sub-mapping (frequency). Here we would take the first position in frequency and go down to its according sub-mapping which is a one dimensional time mapping and has a well defined iterator. When we reach the end of this sub-mapping we go back to the “parent-mapping” and go to its next position in frequency, which again points to a time sub-mapping. After all frequency sub-mappings are handled, the same process is repeated for the next point in the space sub-mapping. This gives a well defined way to iterate over arbitrary dimensional mappings and also enables us to define a “bigger than”-comparison of Argument-objects.

5.4 Mapping Implementations

There are several implementations of the Mapping interface already included in the current version of MiXiM. These implementations actually already support most of the scenarios we can think of. Writing own AnalogueModels or creating own Signals is simply done by using these implementations or sub-classing from them.

The first implementation is the SimpleConstMapping. It is meant to be used as a base class for any ConstMapping implementation which is based on a mathematical formula and therefore can be calculated every time its value at a specific position is needed. The only method that has to be implemented is the getValue()-method. Additionally, key entries have to be defined in order to be able to iterate over the Mapping. An implementation of the getValue()-method takes the values of the passed position and maybe some parameters previously set during initialization to calculate the value of the represented mapping at the position specified.
MultiDimMapping by the passed argument.

The next implementation provided is the TimeMapping class. It is intended to be used for any mapping where time is the only domain. The TimeMapping is implemented by using an extended std::map which is able to interpolate between entries with a given interpolation method to return values for positions for which no values have been set.

The last implementation, MultiDimMapping, is used to represent any Mapping whose domain has more dimensions than time. It uses the nesting-method explained in Section 5.3 meaning it represents a multidimensional mapping by mapping from the key entries in one dimension to a number of sub-mappings which again are mappings from the key entries in their dimension to a number of sub-mappings until the time dimension is reached (which always is the last since its id is zero). The mappings of the time dimension are not mappings to a further sub-mapping but to an actual value. This results in the tree-like structure shown in Figure 8. Each MultiDimMapping instance represents a mapping from one dimension to a number of sub-mappings. These sub-mappings can then be instances of further MultiDimMappings or they can be TimeMappings which represent the leafs of the tree-like structure. The actual mapping from the positions in the current dimension to the sub-mappings is done by the same extended std::map used for the TimeMapping with special interpolation methods which are able to interpolate a whole sub-mapping.

5.5 Mapping Utilities

There are a number of utility methods provided in a MappingUtils class. These methods enable the easy creation and combination of mappings without having to worry about the implementation details.

There are methods which add, subtract, divide or multiply two ConstMapping instances element-wise and return the result in a new Mapping instance. These methods are using the same applyElementwiseOperator()-method which applies a passed operator to the values at every key entry of both of the passed mappings and returns the result in a new Mapping instance. The passed operator can be any operator which takes two double values and returns a new double value, e.g., the std::plus or std::multiplies operator. The set of key entries of the result is the union of the sets of key entries of the passed mappings.

Figure 5 shows an example of the multiplication of 1-dimensional mappings and the resulting key entries. Figure 5(a) shows a power mapping using a different power for the preamble and the rest of the message. Figure 5(b) shows the radio state model of the receiver and Figure 5(c) a simple path-loss model. Note that the path loss model has a lot more key entries than the power and radio state models. Figure 5(d) shows the multiplication result and its key entries. In this 1-dimensional case the resulting mapping is a TimeMapping-instance which interpolates between the union of the key entries of the input methods. If the result is multidimensional an instance of MultiDimMapping is returned.

The applyElementwiseOperator()-method can even operate if the passed ConstMappings are not of the same domain, as long as the domain of the second ConstMapping is a subset of the first one. This is, e.g., useful for using a simple AnalogueModel which only defines an attenuation mapping over time together with a more complex Signal whose power mapping is defined over time and frequency. This would obviously imply the assumption that the attenuation over time defined by the AnalogueModel is identical for every frequency.

An example for the multiplication of a two-dimensional power mapping with a one-dimensional RadioStateAnalogueModel is shown in Figure 9. Figure 9(a) shows the power mapping defined over time and frequency where the dots mark the key entries. Figure 9(b) shows the radio state mapping which is only defined in the time domain. To be able to multiply the one-dimensional radio state mapping with the two-dimensional power mapping the method “copies” the radio state mapping for every frequency the power mapping has key entries in, in this case for frequency of 2.412, 2.422 and 2.432. The result can be seen in Figure 9(c). Then the power mapping and the “filled up” radio-state mapping can be multiplied as usual by multiplying the values at the key entries defined by both mappings. The result is shown in Figure 9(d).

The MappingUtils class also provides a method called createMapping() which takes a DimensionSet and an interpolation method and returns an appropriate Mapping instance (which will be either a TimeMapping or a MultiDimMapping, depending on the passed DimensionSet). So the user doesn’t have to care about correct creation of the Mapping. The available interpolation methods are “STEPS” which interpolates by using the next smaller key entry as value (resulting in a step-like function), “NEAREST” which uses the nearest key entry as value and “LINEAR” which interpolates linearly between the next smaller and the next bigger key entry.

6. CONCLUSION

In this paper we present an architectural overview of the modeling and the implementation of the physical layer within the MiXiM simulation framework. MiXiM is a simulation framework supporting wireless transmissions within OMNeT++. One of the main advantages of the presented approach is that it supports modeling of arbitrary complex signals (in time, frequency and space) but at the same time also supports simple single-frequency, single-(isotropic)-antenna models without carrying the overhead of complex models.

The modular structure of MiXiM enables a high degree of model reuse and the almost arbitrary combination of models. The same simulation can be carried out with a simple path-loss model or with complex shadowing and fading models just by adding / removing the appropriate models in the configuration file.

MiXiM already contains a number of AnalogueModel implementations for path-loss, shadowing, and fading as well as popular standards as IEEE 802.11 and 802.15.4 (ported from the Mobility Framework [2]). However, the main focus so far was on the archi-
7. REFERENCES


