
How To Use *PeerfactSim.KOM* Visualization

Julius Rückert, Leo Nobach

KOM - Multimedia Communications Lab, Document Version 0.4



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

1	Introduction	2
1.1	When should I use PeerfactSim.KOM Visualization?	2
1.2	When should I avoid using it?	2
1.3	What to do to visualize my simulation?	2
2	Hint: Usage of GUIRunner	4
3	Use <i>PeerfactSim.KOM Visualization</i> with preconfigured scenarios	5
3.1	Basic usage	5
3.2	Show a metric for each node	7
3.3	Usage of the Filter	8
3.4	Export gnuplot files	9
4	Key concepts of the visualization backend	10
4.1	Analyzer backend interface	10
4.2	Overlay adapters	10
4.3	Schematic Positioners	11
4.4	Superordinate Multi Positioners	11
4.4.1	The Hierarchical Positioner	12
5	Configuring scenarios to use <i>PeerfactSim.KOM Visualization</i>	13
5.1	Basic configuration	13
5.2	Display generic node interaction	13
5.3	Adding overlay adapters	14
5.4	Advanced: Adding superordinate positioners	14
6	Implementing new overlay adapters	15
7	Advanced concepts	16
7.1	Writing a new transformer (optional)	16
7.2	How to use the bitmap based distribution	18
7.2.1	Let GNP derive host positions from a bitmap	18
7.2.2	Visualize the hosts on the map	19

1 Introduction

1.1 When should I use PeerfactSim.KOM Visualization?

PeerfactSim.KOM Visualization is designed to visualize the state and network activity of the *PeerfactSim.KOM* simulation framework for many purposes, e.g.

1. to help developers of components, especially overlays, to find bugs and to support the development process in general,
2. to show the operation of peer-to-peer networks to students for educational purposes,
3. to present peer-to-peer technology to customers and other interested parties.
4. to measure performance metrics (for small- and medium-sized network scenarios).

If you want to use *PeerfactSim.KOM Visualization* for one of these purposes, it is likely that it will help you.

1.2 When should I avoid using it?

In few cases it is likely that *PeerfactSim.KOM Visualization* will not help you, like

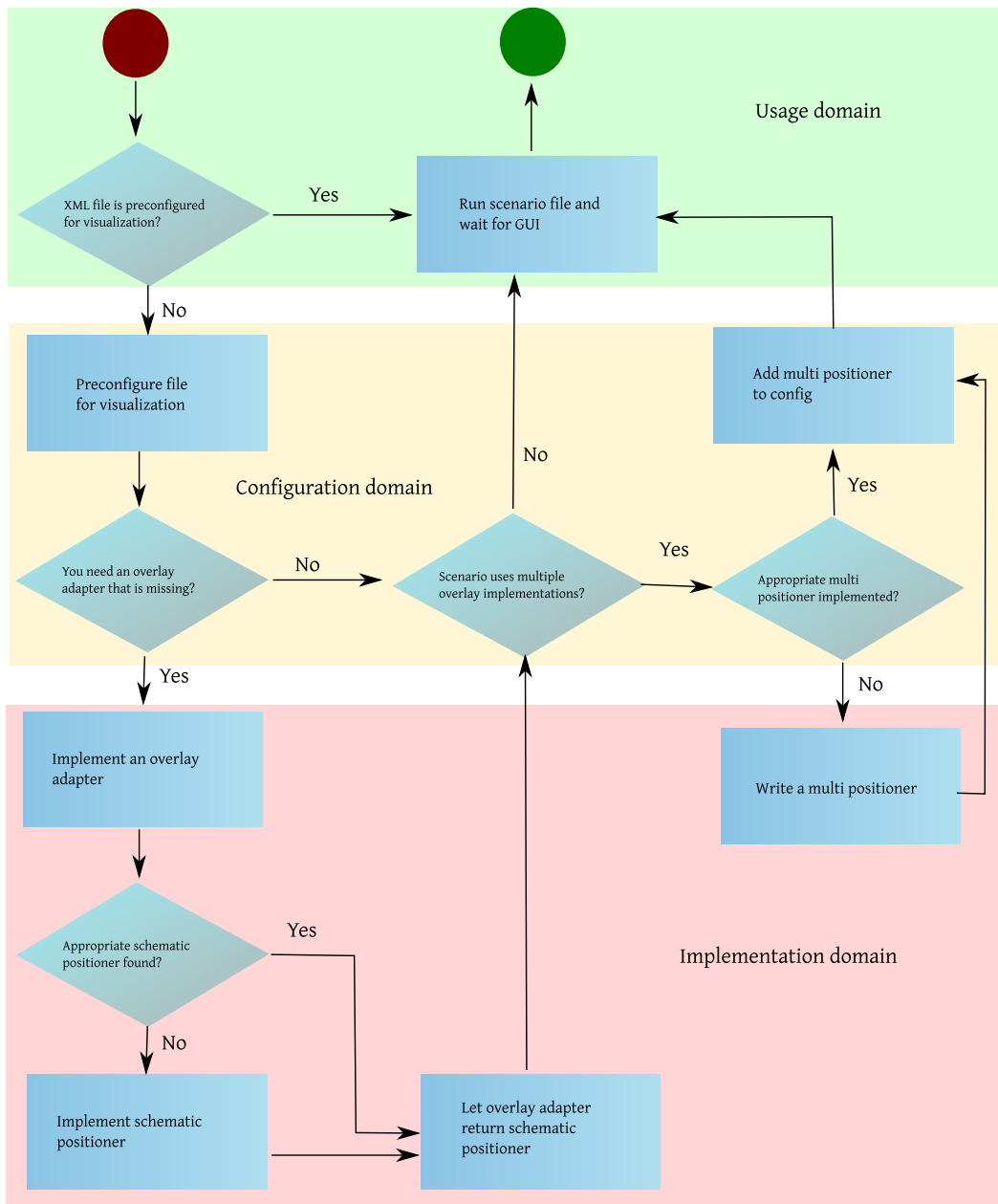
1. to try out scalability bounds of a network in terms of **peer count** and **message count** if you require a fast image rendering rate. *PeerfactSim.KOM Visualization* was developed to show each quantum of network activity to the user, like single messages and operations. Configuring scenarios bigger and even bigger would render an unreadable image and its generation process would consume a lot of realtime performance. Testing results show that an average scenario greater than 80 nodes will result in a maximum image rendering rate less than 1 image per second on a common workstation (Centrino Duo, 2GB RAM) and a CPU fan making a lot of noise.

Exception: You can still use the *Gnuplot Export* function. There are no realtime requirements for it, so there is no node/message count limit for good performance. But prepare to wait a little bit for the result to show up.

1.3 What to do to visualize my simulation?

First, make sure if your configuration file (one of the XML files in the directory `config`) is preconfigured for visualization. Follow the chapter *Configuring scenarios to use PeerfactSim.KOM Visualization* how a preconfigured file looks like and configure it, if necessary. For convenience, preconfigured files should be placed into the subdirectory `config/visualization`.

For configuration of this file, you possibly need overlay adapters. The chapter *Implementing new overlay adapters* will tell you how to do this.

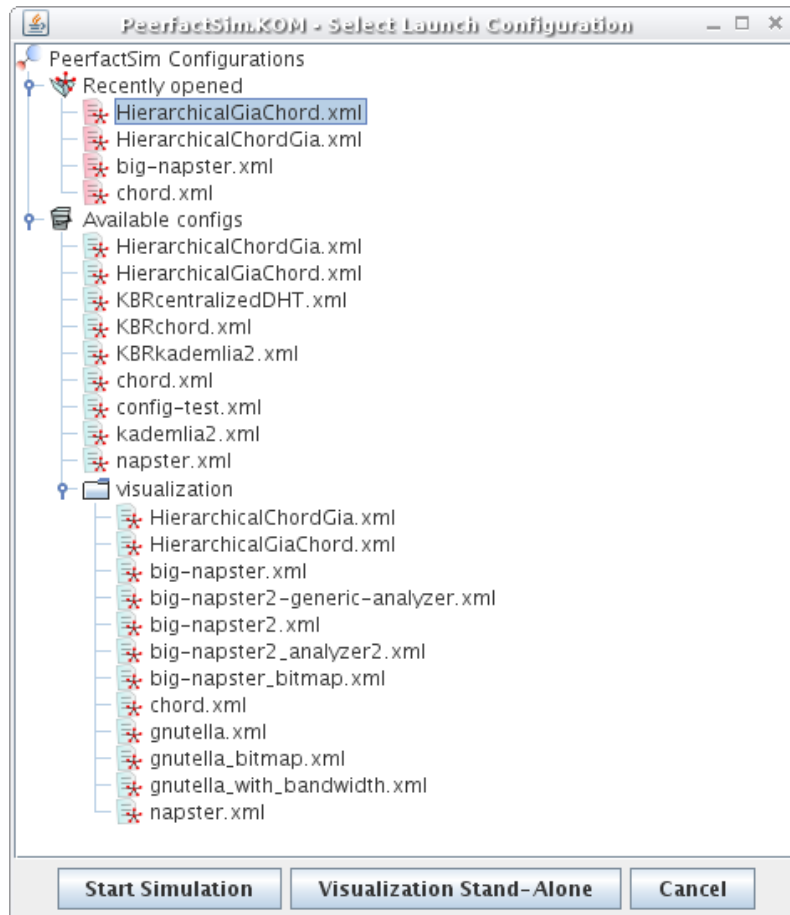


An activity diagram that I suggest for getting your scenario configuration visualized.

Don't be scared by the terms used in here. They will be explained in the following chapters. Note that you do not always need an overlay adapter for the visualization of an overlay. Read the chapter *Configuring scenarios to use PeerfactSim.KOM Visualization* for more on that.

2 Hint: Usage of GUIRunner

You can run *PeerfactSim.KOM* using a GUI. Simply run the class `de.tud.kom.p2psim.GUIRunner`. The window that allows you to select available configuration files will help you as a beginner as well as a professional to quickly run the configuration scenario you want.



The window will show a tree of recently opened and other available configuration files in the directory `/config`. Just double-click on one of them to run the preconfigured scenario in *PeerfactSim.KOM*. The simulation will start, and if *PeerfactSim.KOM Visualization* is configured for this scenario, the visualization component will pop up.

As you can see there is an option to run the visualization stand-alone to load recordings.

3 Use PeerfactSim.KOM Visualization with preconfigured scenarios

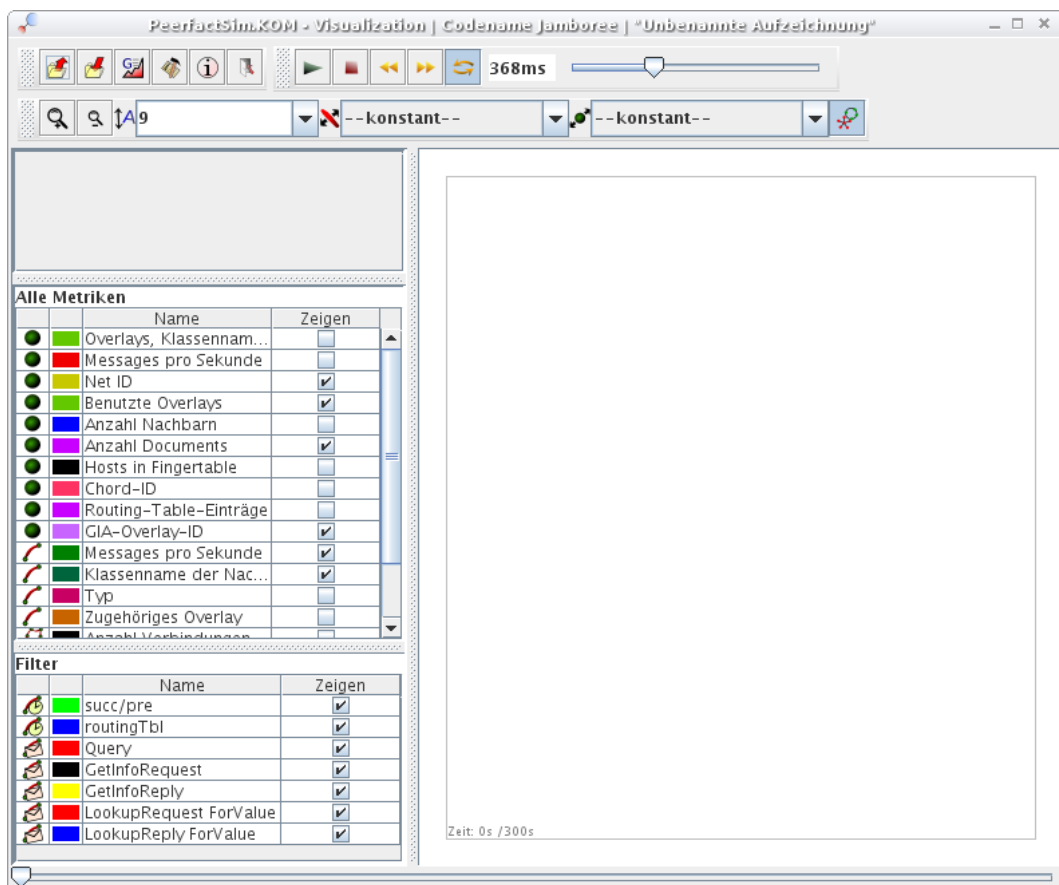
A scenario needs to know how it is supposed to be visualized. This means that each scenario configuration in PeerfactSim.KOM needs to have a visualization-specific part. If you want to preconfigure an unconfigured scenario, please read the following chapters how to do this.

For convenience, all preconfigured scenario XML files are placed in `/config/visualization`.

Simply run PeerfactSim.KOM with a preconfigured scenario. I recommend GUIRunner, but feel free to run it via command line e.g. The simulation will start and the visualization window will pop up, if no exception was thrown during the simulation process.

3.1 Basic usage

The main components of the GUI are designed like in a basic multimedia player. Play, Pause, Stop, Forward, Reverse, the slider on the bottom are derived from common player GUIs. Just try it out.



The main component after startup

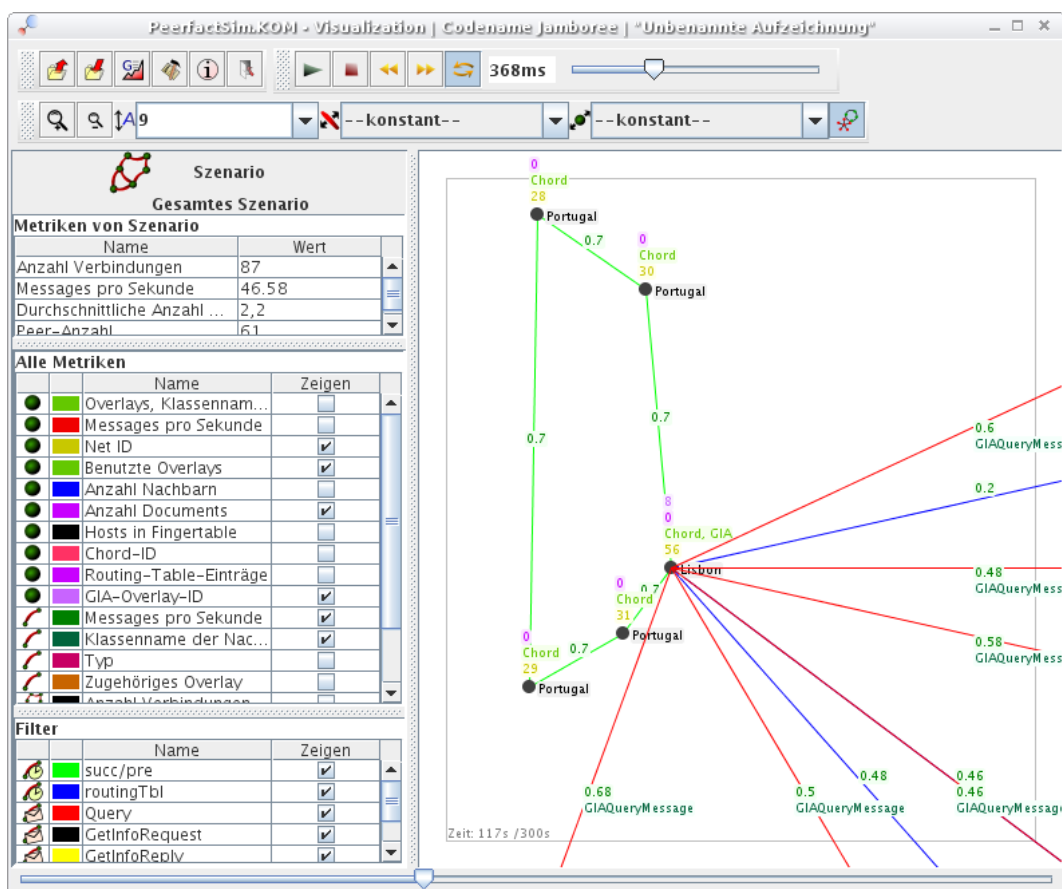
Further well-known commands are:

1. Save your visualized scenario to disk by pressing the *Save to File* button.
2. Load your scenario from disk by pressing *Load from File*.

You can use your mouse (or keyboard) in the visualization window to modify your view:

1. Zoom in using your mouse wheel (pgUp/pgDn).
2. Drag the view to a position while left-clicking into it (arrow keys).

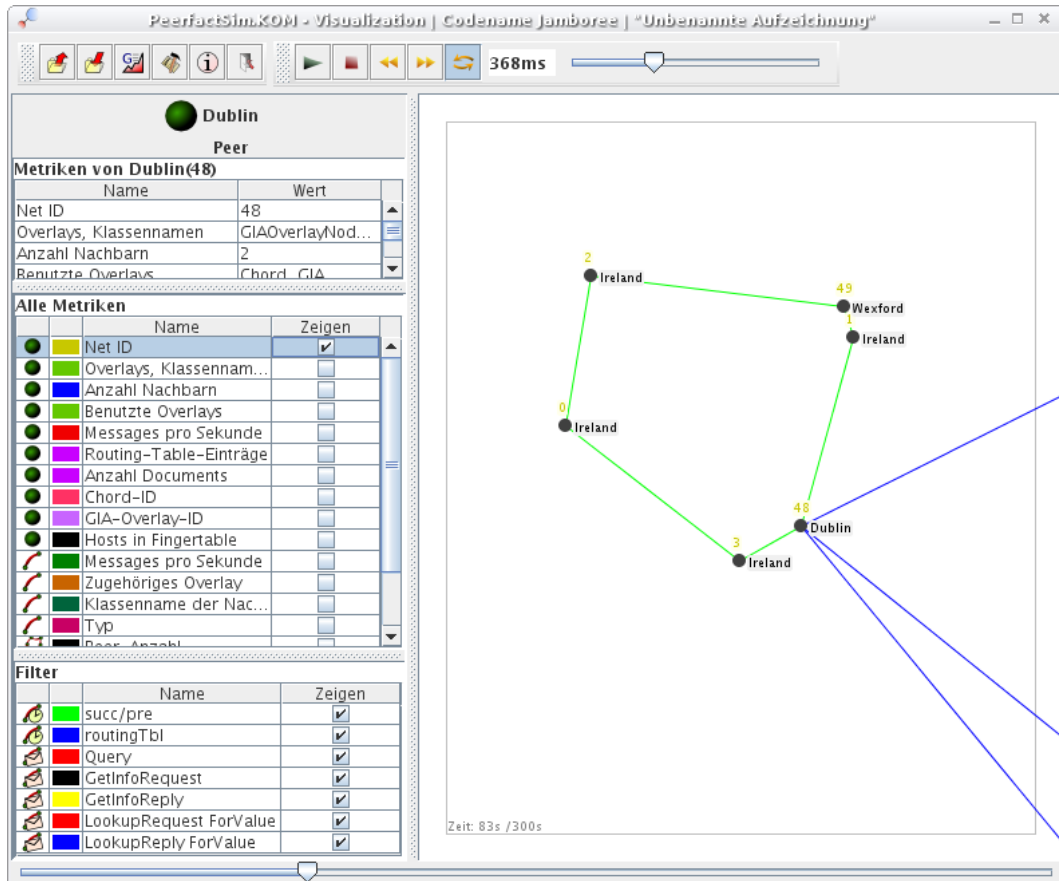
Displaying metrics and attributes of a single entity: Click on a node or an edge. You will see all of its attributes and metrics on the side panel. By clicking into empty space, you will see the attributes of the whole scenario.



The main component during a scenario running

3.2 Show a metric for each node

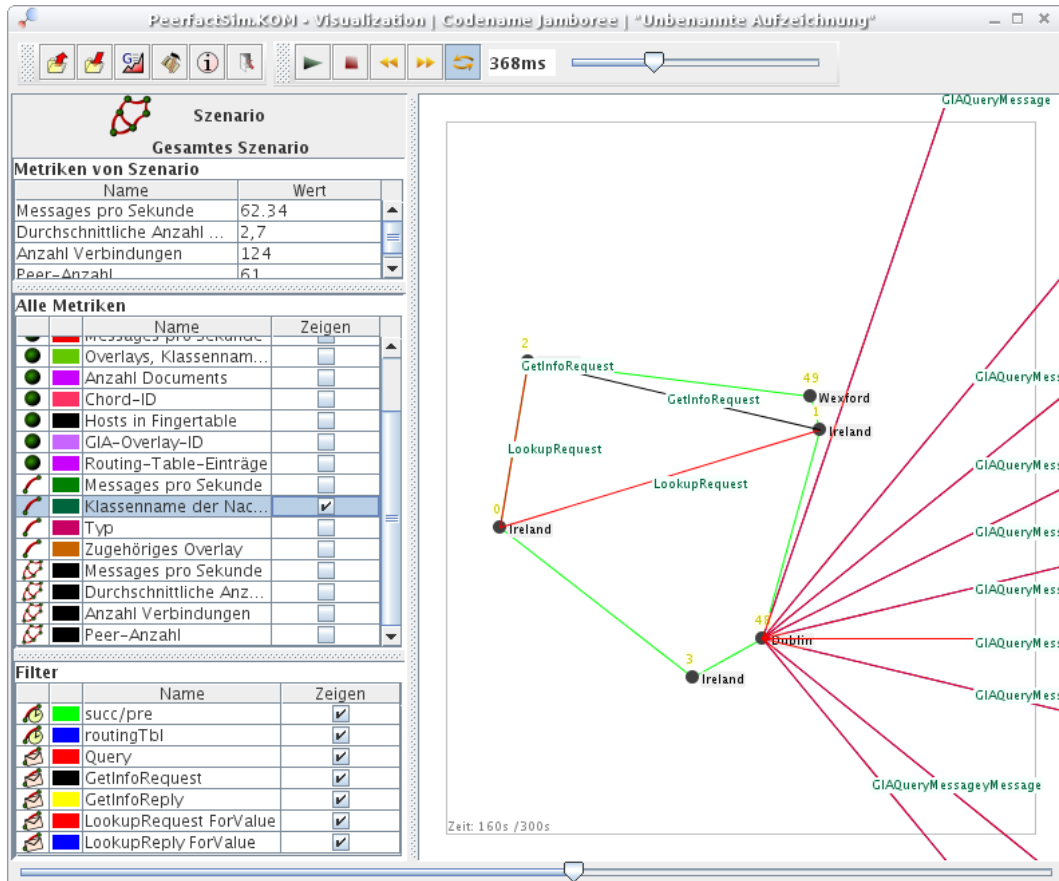
The metrics panel is located at the sidebar. This is a list of all available metrics in the scenario, splitted up in metrics of nodes, edges and scenario metrics. Just select/deselect one or more of them (check box). Selected metrics will show up directly on the entities in the graphical view.



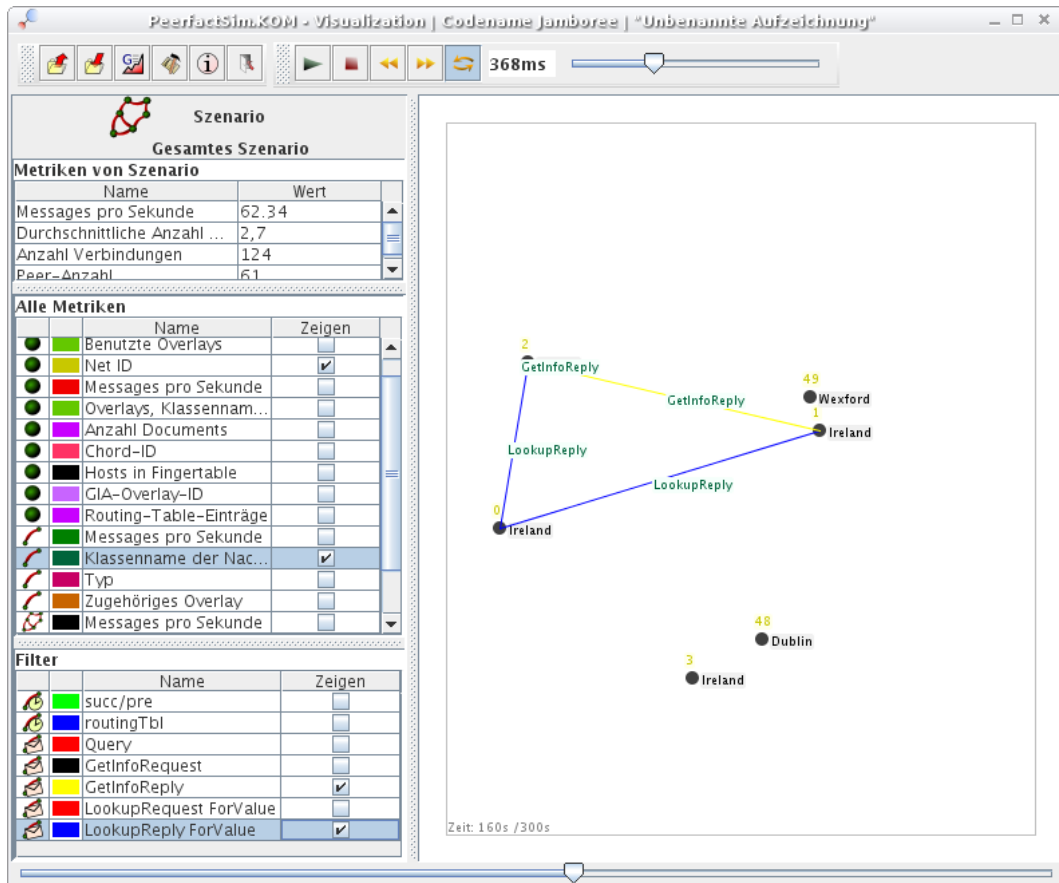
Metric "NetID" selected in main component

3.3 Usage of the Filter

The Filter's purpose is to disable the rendering of connections or messages that are irrelevant for visualization or disturbing it. For example if you don't want to show all download requests, but only routing table references in a network, just select the events you want to see.



All edges visible



All edges filtered except for Replies

3.4 Export gnuplot files

Select the *Export to Gnuplot* button to generate a gnuplot from the scenario. Select one of the possible modes in the tabs:

1. Select the entities (either nodes or edges) you want to compare to each other, select which metric to use as value.
2. Select the metrics of the currently selected object to export its metrics to Gnuplot.

Note that only numerical metrics can be visualized as a graph, so only these ones are listed here.

4 Key concepts of the visualization backend

This chapter is an introduction to the concepts of the backend to generate visualization-relevant events. Knowledge about the concepts is required if you are writing overlay adapters or positioners, and it is recommended for configuring scenarios. You do not need to read this if you only want to use preconfigured scenarios.

4.1 Analyzer backend interface

The visualization has a backend implementing the interface `Analyzer`¹ of `PeerfactSim.KOM`. Its purpose is to extract visualization-relevant events and convert them into a viewable format that is put into an event timeline. However, it needs help by classes that have to be implemented for each overlay in order to extract overlay-specific events.

4.2 Overlay adapters

An overlay adapter collects overlay-specific visualization information out of the runtime behavior of an overlay implementation. This includes the following things:

1. If a host is joining the scenario, it is able to add attributes to the node that describe its overlay-specific behavior (e.g. its role in the overlay like server or client, or the document count).
2. If a visualization-relevant event occurs (maybe caught by an event listener or continuous state check on each message), it changes the attribute value at the node of it to the new value (e.g. a host's document count has been increased). and puts this event onto the timeline.
3. It creates coloured connections between nodes if nodes somehow are *connected* to each other, according to the definition of *connection* of an overlay (e.g. the successor/predecessor relation in Chord).
4. It creates *flash events*², like flash edges, if an important message has been exchanged between nodes.
5. It defines, which schematic positioner shall be used to set a position for a host implementing the overlay of the adapter (e.g. a **ring** in Chord).

The overlay adapters register for each overlay implementation (in most cases only one) and each event (message, operation) they are interested in. They will be called by the analyzer if the event occurs.

¹ The subinterfaces `OperationAnalyzer`, `TransAnalyzer`, `NetAnalyzer`, `ConnectivityAnalyzer` of `de.tud.kom.p2psim.api.analyzer.Analyzer`

² Events without a relevant duration that shall only be shown a short time

4.3 Schematic Positioners

A positioner is an object that defines a schematic position for a specific host.

In the visualization GUI you can toggle between net position and schematic position. The net position is defined by the host itself. The schematic position has to be determined by a positioner of the *PeerfactSim.KOM Visualization* .

Its function is decoupled from the overlay adapter as there can be many instances of positioners, one instance per overlay network of an overlay implementation, for example.

There are some generic positioners placed in `de.tud.kom.p2psim.impl.util.vis.analyzer.positioners.generic` you can use. If no positioner is given by the overlay adapter (null) the positioner defaults to `FCFSPartitionRingPositioner`.

4.4 Superordinate Multi Positioners

What if there is a node that implements two overlays? Which positioner to take to determine its schematic position? The default behavior is to take the first overlay given in the scenario configuration to determine the position. Only if the positioner is null or can't deliver a position, the second overlay adapter's positioner is used and so on.

You can alter these semantics by selecting another multi positioner.

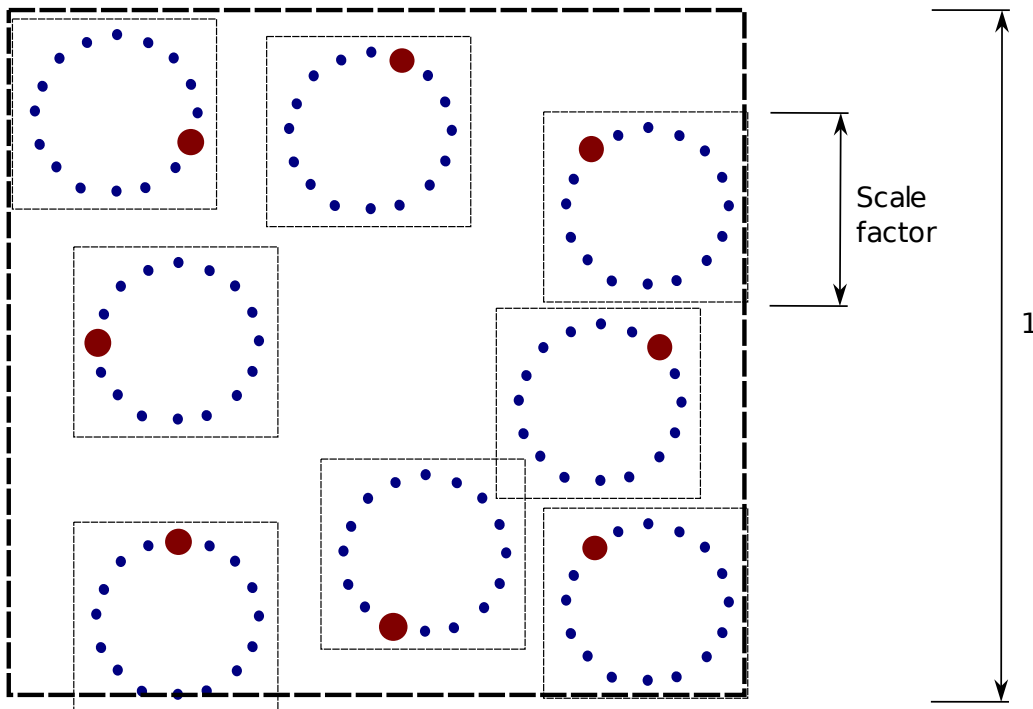
4.4.1 The Hierarchical Positioner

This positioner is thought for some kind of hierarchical networks, in which a supernet using an overlay implementation connects many supernet nodes, and each of the supernet node connects many of the subnet nodes using another overlay implementation. Examples for these networks are GIChord and ChordGIA, here the first name indicates the supernet, the second name the subnet to be used.

The result of the supernet's positioner



The result of the positioners assigned to each subnet



The result of the multi positioner using these examples.

Concept of the hierarchical positioner

5 Configuring scenarios to use *PeerfactSim.KOM Visualization*

5.1 Basic configuration

A scenario can be configured for *PeerfactSim.KOM Visualization* by adding the visualization analyzer¹ to it like in the following example.

```
<Monitor class="...">
  <Analyzer class="de.tud.kom.p2psim.impl.util.vis.analyzer.VisAnalyzer"/>
</Monitor>
```

This example will display hosts joining and leaving the scenario, but no interaction between the nodes themselves, like messages they pass to each other, overlay-specific internal states etc.

5.2 Display generic node interaction

To extend the displaying of nodes with interaction between them, you can try out a special feature. Just add the XML attribute `messageEdges="true"` to the analyzer tag.

```
<Monitor class="...">
  <Analyzer class="de.tud.kom.p2psim.impl.util.vis.analyzer.VisAnalyzer"
    messageEdges="true"/>
</Monitor>
```

This will paint an edge for each message between nodes. The metric "Class name of message" will show you the Java class name of the subtype of `Message`² sent between nodes.

Note that this is only a workaround for overlays with unimplemented `OverlayAdapters`. The result could lead to a flood of irrelevant messages between hosts (like ping/pong messages in GIA), depending on the complexity of the overlay implementation. In this case use the edge filter or write an overlay adapter (explained in the further chapters), the latter will be the most expressive method, but it requires some lines of code to be written.

¹ `de.tud.kom.p2psim.impl.util.vis.analyzer.VisAnalyzer`

² `de.tud.kom.p2psim.api.common.Message`

5.3 Adding overlay adapters

Overlay adapters allow the implementation of overlay-specific visualization behavior. They are located in the classpath `de.tud.kom.p2psim.impl.util.vis.analyzer.ol` and subclass `de.tud.kom.p2psim.impl.util.vis.analyzer.OverlayAdapter`. How to write overlay adapters will be explained in the further chapters.

```
<Monitor class="...">
  <Analyzer class="de.tud.kom.p2psim.impl.util.vis.analyzer.VisAnalyzer">
    <OverlayAdapter class="[Class]">
    <OverlayAdapter class="[Class 2]">
    <OverlayAdapter class="[Class 3]">
    ...
  </Analyzer>
</Monitor>
```

Simply add them to the Analyzer tag (Substitute "[Class x]" with the classpath of your overlay adapter, of course). Most scenarios only need one analyzer to be added, but feel free to add one overlay adapter for any overlay implementation your scenario holds (if available).

Notice: The ordering of the adapters in the XML document is important! If no multi-positioner is added, the semantics default to this: The schematic position of the first node is defined by the first overlay adapter, if he cannot supply a position for it, the second adapter is asked for and so on.

5.4 Advanced: Adding superordinate positioners

In some situations it is relevant to use superordinate multi-positioners handling different positioning concepts to achieve a special goal. Just add them to the monitor.

```
<Monitor class="...">
  <Analyzer class="de.tud.kom.p2psim.impl.util.vis.analyzer.VisAnalyzer">

    <MultiPositioner
      class="de.tud.kom.p2psim.impl.util.vis.analyzer.positioners.multi.HierarchicalPositioner"
      scaleFactor="0.13f"/>

    <OverlayAdapter
      class="de.tud.kom.p2psim.impl.util.vis.analyzer.ol.chord.ChordAdapter"/>

    <OverlayAdapter
      class="de.tud.kom.p2psim.impl.util.vis.analyzer.ol.GIA.GIAAdapter"/>

  </Analyzer>
</Monitor>
```

In this example the hierarchical positioner is added to the analyzer. If you set it, the semantics of the following overlay adapters will be changed: The first overlay adapter is treated as the subnet, the second one as the supernet connecting all the subnets. The scale factor parameter (special parameter, only for Hierarchical Positioner) is used for scaling the subnets to make them smaller than the super net and let them fit into the picture. Please look into the Javadoc for special parameters for your superordinate positioners.

6 Implementing new overlay adapters

Create a class file that extends `OverlayAdapter`¹ and implement its missing methods. For convenience, you should put this class and further classes of your overlay adapter into a new subpackage of `de.tud.kom.p2psim.impl.util.vis.analyzer.ol`.

The abstract subclass `OverlayAdapter` is extensively documented, so you should have no problems understanding the purpose of the methods. However, there is an example Java source file² that you should consider for your development process. It uses example commands for manipulating visualization events and explains their effect.

Monkey see, monkey do: If the information in this example class is not enough, try to look into existing overlay adapters, like Napster, Chord or GIA how they are implemented and how they extract visualization information out of the overlay implementations. They should be well-documented.

¹ `de.tud.kom.p2psim.impl.util.vis.analyzer.OverlayAdapter`

² `de.tud.kom.p2psim.impl.util.vis.analyzer.ol.example.ExampleOverlayAdapter`

7 Advanced concepts

7.1 Writing a new transformer (optional)

The following steps are not necessary if a transformer for the type of `NetPosition` used in your scenario was already implemented. To check this take a look at the package “`de.tud.kom.p2psim.vis.adapter.analyzers.transformNetPositions`”.

If there is no transformer for the special kind of `NetPosition` that you need, create your own one. To explain the general proceeding for this, let’s have a look on an simple example: the class “`GnpPositionTransformer`”.

The whole class looks like this:

```
package de.tud.kom.p2psim.vis.adapter.analyzers.transformNetPositions;

import de.tud.kom.p2psim.impl.network.gnp2.topology.GnpPosition;
import de.tud.kom.p2psim.vis.util.visualgraph.Coords;

public class GnpPositionTransformer implements INetPositionTransformer<GnpPosition>{

    @Override
    public Coords transform(GnpPosition netPos) {

        // Here the tranformation takes place
        float x = new Double(netPos.getGnpCoordinates(0)).floatValue();
        float y = new Double(netPos.getGnpCoordinates(1)).floatValue();

        Coords coords = new Coords(x,y);

        return coords;
    }
}
```

This transformation is very simple. There is only one method. On invocation of this method we are given an instance of `GnpPosition` and have to decide how to extract an 2-dimensional representation of this position. In this case we just pick out the first two coordinates and convert them to `Float`. In other cases this could be a more difficult computation.

After writing the transformer you have to register it, so the visualization can use it. Therefore open the class “`de.tud.kom.p2psim.vis.adapter.analyzers.AVisAnalyzer`” and take a look at the method with the signature “`public Coords transformPosition(NetPosition netPos)`”. There you will find the following lines of code:

```
// Is it a SimpleEuclidianPoint?
if(netPos instanceof SimpleEuclidianPoint){
    coords = new SimpleEuclidianPointTransformer()
        .transform((SimpleEuclidianPoint) netPos);

// Is it a GnpPosition?
```

```
} else if(netPos instanceof GnpPosition){
    coords = new GnpPositionTransformer()
        .transform((GnpPosition) netPos);

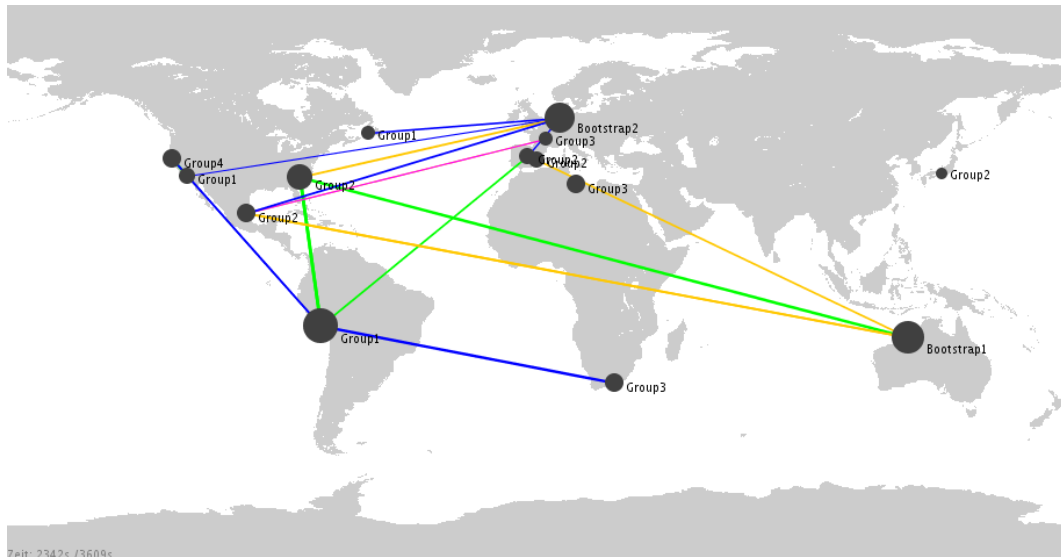
// Is the type not supported?
} else {
    coords = new Coords(0,0);
    System.err.println(this.getClass().getName()+
        " - There is no transformer for
        the given type of NetPosition.");
}
```

To support the new type of NetPosition you have to add another “else if”-branch according to the existing ones. After this steps, the new NetPosition type can be used and hosts should be placed the right way on the visualization window.

7.2 How to use the bitmap based distribution

In the following section I will describe how you can use the bitmap based distribution to create nice looking visualizations based on a map. For this purpose we use a special mode of the GNP network to derive host positions from a bitmap instead of using a `gnp_file`. After that we choose a picture for the background inside the visualization interface.

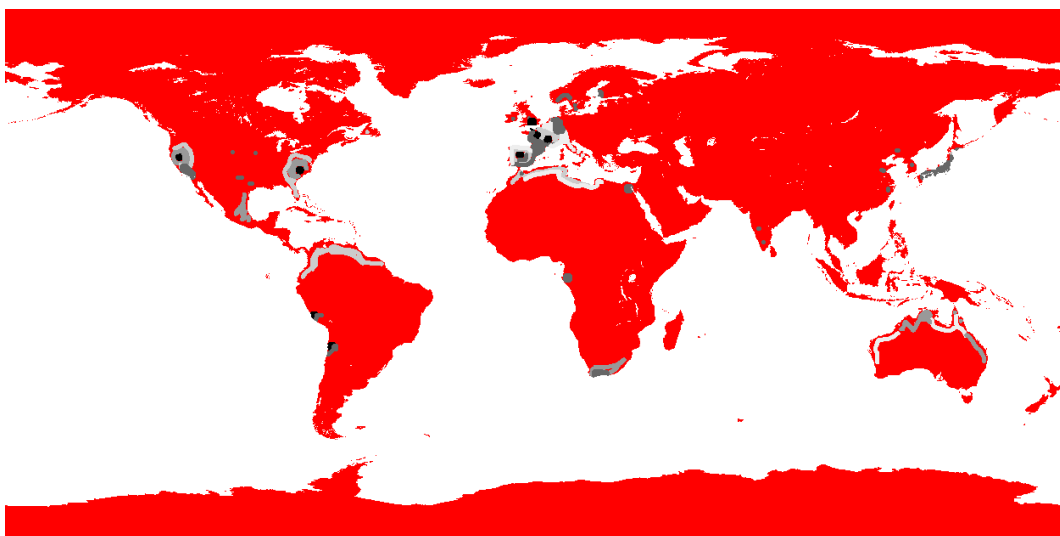
The resulting visualization could then look like this example:



7.2.1 Let GNP derive host positions from a bitmap

For this step it is important to use a 8-bit bitmap. You can create such a bitmap with hardly every picture editing software by saving the bitmap in a mode where only 256 colors are used.

The distribution of hosts on the map is derived from the grey values of the used bitmap. Positions with darker color are used with a higher probability as host positions than positions with lighter colors. A nice distribution on a worldmap could be achieved by the following bitmap:



In order to use your own bitmap you have to copy it into a folder that the simulator is able to access (I chose the config folder). Then you have to configure GNP to use the bitmap by editing the definition of the netlayer in the config file. The result could look like these lines:

```
<NetLayer
  class="de.tud.kom.p2psim.impl.network.gnp.GnpBitmapNetLayerFactory"
  downBandwidth="50" upBandwidth="25" PbaPeriod="1"
  experimentSize="$size" bitmapPath="config/worldmap.bmp" >
  <LatencyModel
    class="de.tud.kom.p2psim.impl.network.gnp.GnpLatencyModel" />
  <BandwidthManager
    class="de.tud.kom.p2psim.impl.network.gnp.GnpNetBandwidthManagerPeriodical" />
</NetLayer>
```

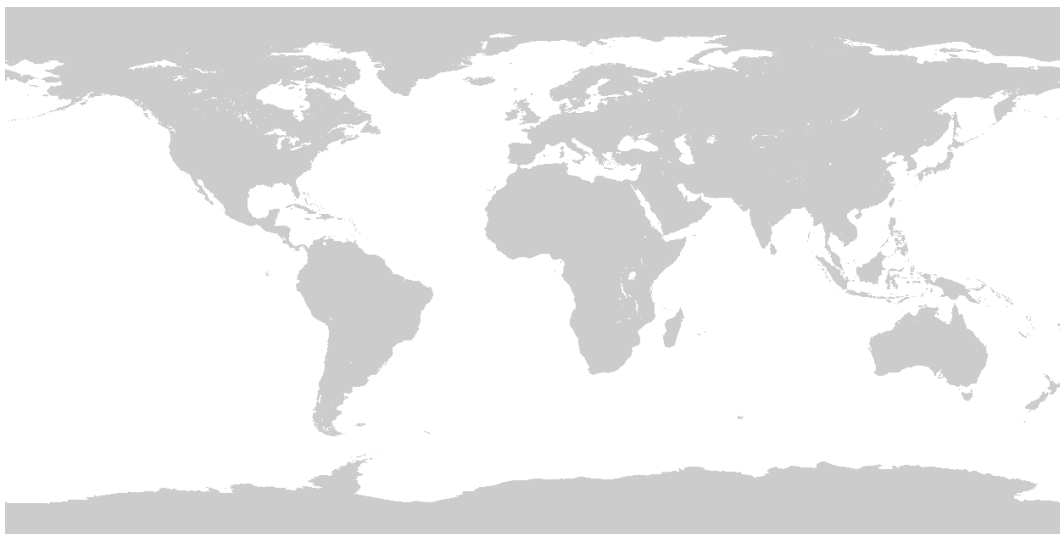
The important differences to the use of the normal GNP network are that we call the factory “GnpBitmapNetLayerFactory” instead of “GnpNetLayerFactory”. This factory does not support all attributes the normal GNP supports. Therefore these attributes have been removed and a new attribute “bitmapPath="config/visualization/worldmap.bmp"” was added.


After this changes your simulation will use positions generated on base of the chosen bitmap.

7.2.2 Visualize the hosts on the map

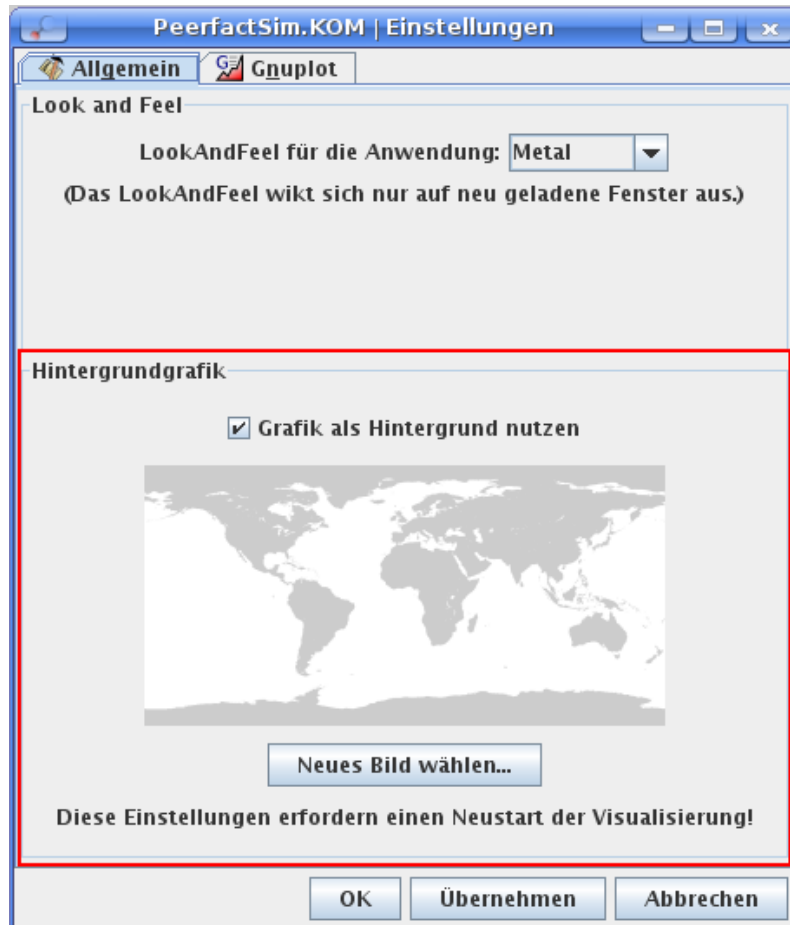
In this step I assume that you already connected the visualization to the simulator. The visualization interface shows up when launching the simulation.

What you have to do next is to choose the picture you want to use as background for the visualization. This picture should have the same dimensions as the bitmap you used for the generation of the positions. It does not need to include the different grey scale information the simulator needs to generate the distribution. In case of a map I recommend to use two different pictures. The bitmap to generate positions and a second picture that only includes the information you need for an understandable visualization. I chose the following one for the visualization:



You can choose the background image within the configuration window that shows up when you click the button .

The configuration window shows up:



Here you can choose a background image and enable/disable the use of it.
(IMPORTANT: After this changes you have to restart the simulation)