# Intro

First, we decided that our application should be scalable upon a big number of clients. Therefore we first thought a lot about possible network topologies. A fully-connected graph would be easy to implement, but would not met our requirements for a scalable application. We also decided against a pure UDP-broadcast solution because of the possible package loss. Our decision fell on a spanning tree topology where packages are routed and flooded efficiently.

Our second decision we made upon the fact that we were not sure whether an existing framework could realize our desired network topology or not. Thus we decided that we won’t use any existing framework and rely on basic TCP sockets.

Concerning the structure of the document, we decided not to partition the document into a higher level structure, e.g. per line or sentence. Therefore our text document is a simple string (-builder). Text changes, which are sent in the network, only contains the position in the text and the change itself. A change can include a string which will be inserted on a specified position, or a number of characters which should be deleted at a specified position.

# Architecture

Since we weren’t sure which network-framework would fit our need best, we designed our architecture with a flexible network layer. Therefore we specified a network layer which defined only interfaces and common network logic. On top of the network logic the document-layer defines the business logic for managing the document, the curser positions of the clients, etc. Finally, the editor-layer, which uses the WPF, displays the document text, and handle user input.

The following sections contains a more detailed description of each layer.

## Editor-Layer

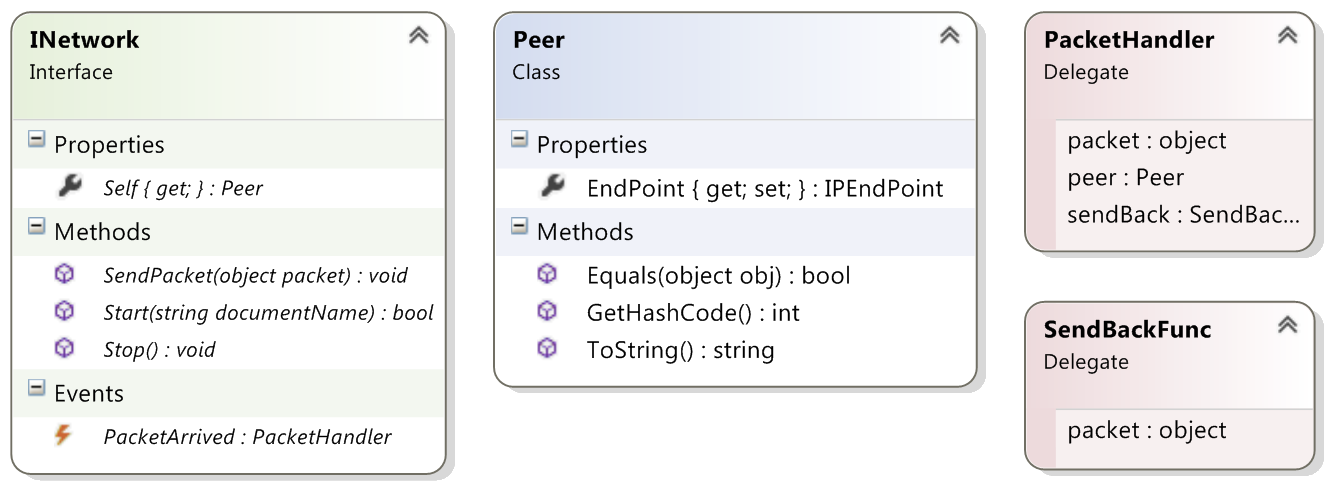
The Editor only contains a single window (MainWindowView) and its corresponding ViewModel. Within the ViewModel everything we display in the GUI is stored. Changes made by the GUI are reflected to the view model, which are then forwarded to the document-layer. A customized TextBox handles the coloring of different regions of the text.

## Document-Layer

The document layer specifies a document-interface for connecting to a document, submit text changes, and events for text- and cursor changes. Changes to the document caused by the UI (Editor layer) are converted into network packets and given to the network layer. Incoming packets from the network layer are integrated into the document’s state and usually trigger events which update the UI. The document layer therefore serves the purpose of a business logic and the data model by interpreting incoming packets and UI events as well as perform operations on the document data itself.

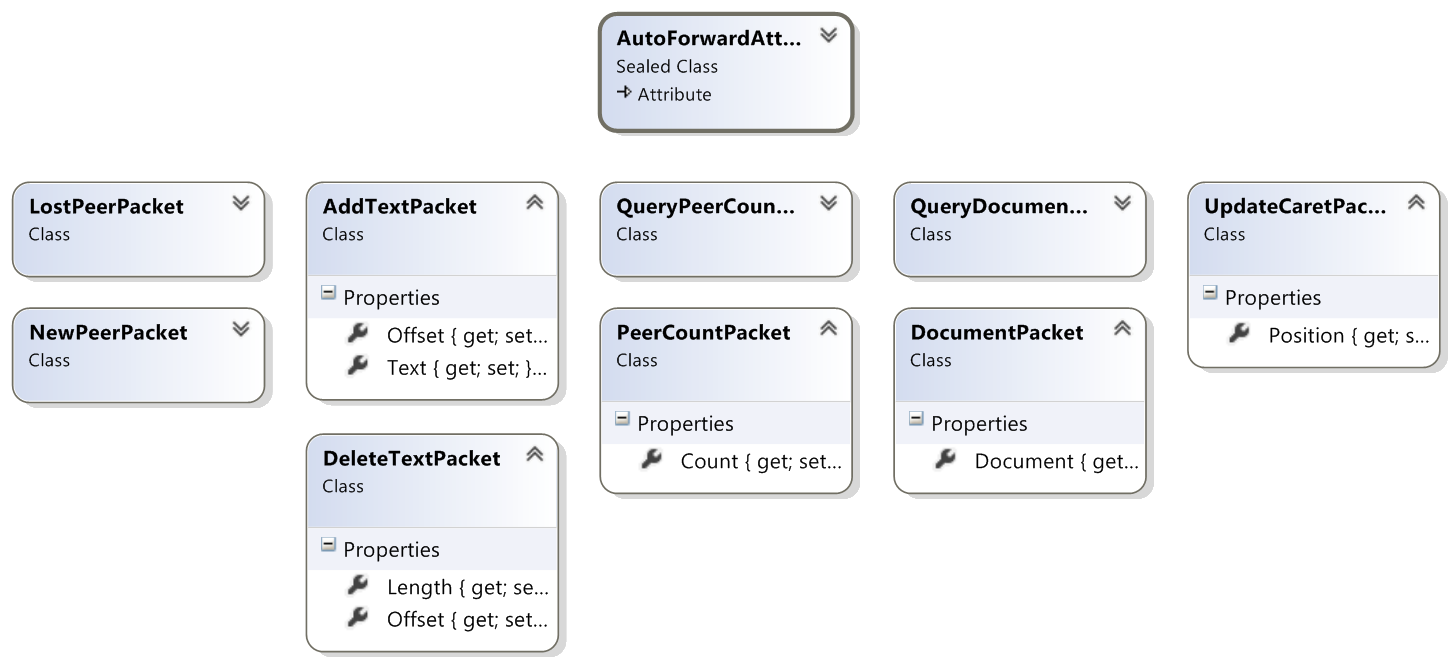
## Network-Layer(s)

The stack of network layers is responsible for managing connections between participants (called peers) of the same document. The network layer is itself structured into several further layers to achieve a clean software architecture. The very top layer contains a C# interface which is injected into the document layer and offers the following four interactions:



* Start(): Initializes the network layer and triggers construction of the network topology. This method requires the document name which is used as an identifier on the network to find peers of the same document. The return value of Start() indicates whether a peer could be found (true) or the current application instance is the first peer for the given document name.
* Stop(): Shuts the network down. Disconnects from all peers of the network topology. After a call to this method no further packages can be sent or are received.
* SendPacket(): Sends a packet into the network. Although packets are themselves classes and all packet types used by the document layer are defined, this method accepts any object. The object passed to SendPacket() is serialized into a binary representation which is then transmitted to all (or one) peers of the network. Generally (apart from forwarding and custom packets) the network layer does not interpret the objects passed to this interface.
* PacketArrived: Event which is raised whenever a packet arrives from the network. This packet can again be any object. In addition to the packet data (object) an instance of Peer is delivered, which serves as a unique descriptor of a peer on the network. The provided sendBack delegate enables higher layers to directly respond to an individual peer as opposed to the SendPacket() method which does not allow the specification of a target.
* Self: Retrievable property holding an instance of Peer which describes the own node/peer on the network. This property is mostly used by the network layer itself to identify the current instance to other peers on the network.

On the highest network layer several kinds of (business logic) packets are available to describe changes to the document and the environment:



* NewPeerPacket/LostPeerPacket to inform other peers on the network that a peer has been added to the network or connection has been lost,
* The AddTextPacket/DeleteTextPacket for document text changes,
* *QueryPeerCount*/*PeerCountPacket* for querying the current number of participants on the network for the set document name,
* *QueryDocumentPacket*/*DocumentPacket* for querying and receiving the whole document text and
* UpdateCaretPacket for updating the caret position.

The optional attribute AutoForward can be declared on any packet and specifies that the packet should be forwarded to all peers of the network or only to a certain destination. Packets names in italic do not have the AutoForward attribute and are typically used to query information from a single peer.

## Network.Tcp.SpanningTree-Layer

TODO

## Network.Tcp.CompleteGraph-Layer

TODO

# Network Algorithms

The algorithms and concepts described in the following sections refers to our TCP-SpanningTree-Layer implementation.

## Topology

When connecting to the network, a client connects to a single “parent” client. This way, the clients form a spanning tree as each client connects to another one except the first client. Packages forwarded into the network are not able to be forwarded infinitely because of the lack of cycles in the topology. Because we have n-1 connections established between n clients, it is not possible to form a cycle.

## Connect to Network

When connecting to a network, the client sends a broadcast, looking for other clients with the same document. TODO which client connects to which client?

## Send Packages

## Connection Loss

# Concurrency mechanisms

The network and are not actively using any concurrency mechanisms, as we never experienced concurrency issues. This might come from the fact that most actions are forwarded and reflected to the view model in the GUI and the WPF-bindings schedules changes and events correctly into the GUI-thread.

# Difficulties and Known Bugs

The management of connections, especially the repair mechanism turned out to be very difficult to implement.

We started developing the client as a pure network-application and therefore never expected to run multiple client on the same machine. Because we initially used the IP-address and a well-known port for our application, the changeover to a different identifier and a flexible port was rather troublesome.

# Performance

Our main performance issue emerges not from network or synchronization overhead. Instead it turned out that our custom Textbox, which highlights the locked regions near other editors’ cursors, is very inefficient. The rendering procedure takes quite long and we weren’t able to fix it.

Although, network traffic and overhead cannot be neglected. Packages only contains small changes to the document, a lot of small packages are sent into the network. Collecting changes over some time and sending one big package with all gathered information could reduce some overhead.

Zusätzlich zur Implementierung soll ein etwa 3-4 seitiger Bericht geschrieben werden, der zumindest folgendes beschreibt:

 Beschreiben Sie Ihre Implementierung. Welche Entscheidungen wurden getroffen und warum?

 Wie sieht ihre Lösung in Bezug auf Concurrency, Zugriffskoordination und Synchronisierung aus? Warum haben Sie sich für die gewählten Lösungsansätze entschieden? Welche verschiedenen Ansätze haben Sie ausprobiert und mit welchem Ergebnis?

 Für welches Kommunikationsframework haben Sie sich entschieden und warum?

 Wie viele gleichzeitige User schafft Ihre Lösung? Welche Probleme/Performanceeinbußen haben Sie dabei beobachtet?

 Auf welche Schwierigkeiten sind Sie gestoßen?

 Wieviel Zeit haben Sie wofür gebraucht?