Abschlussprüfung Winter 2020/21

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

TCP/IP Logging Service

Datenbankgestütztes Logging-Tool für TCP- und UDP-Datenübertragungen

Bearbeitungszeitraum: 01.10.2020 – 16.10.2020

Prüfungsbewerber:

Sascha Sörries

[redacted]

Prüflingsnummer 94043

Prüfungsausschuss FIAE Han 8 108

Bildungsträger:

PROFIL GmbH

Gerhardtstraße 3

30167 Hannover

Praktikumsbetrieb:

ERP-Plus GmbH

Georgstr. 22

30159 Hannover

Inhaltsverzeichnis

1. Einleitung 1
1.1 Projektumfeld 1
1.2 Projektziel 1
1.3 Projektbegründung 1
1.4 Projektschnittstellen 1
1.5 Projektabgrenzung 1
2. Analyse 1
2.1 Ist-Zustand 1
2.2 Soll-Zustand 2
2.3 Wirtschaftlichkeitsanalyse 2
2.3.1 Make or Buy-Entscheidung 3
2.3.2 Projektkosten 3
2.4 Projektnutzen 3
3. Planung 3
3.1 Projektphasen 3
3.2 Ressourcenplanung 4
3.3 Vorgehensmodell für das Projekt 4
4. Entwurf 5
4.1 Zielplattform für das Projekt 5
4.2 Datenschutz 5
4.3 Entwurf des Entity Relationship Diagramms 5
4.4 Entwurf des relationalen Datenmodells 6
4.5 Entwurf der GUI für das Konfigurationstool 6
4.6 Entwurf der Programmlogik für das Konfigurationstool 6
4.7 Entwurf des Services 7
5. Implementierung 9
5.1 Versionsverwaltung 9
5.2 Implementierung der Datenbank 9
5.3 Implementierung des Konfigurationstools 9
5.4 Implementierung des Logging-Tools bzw. Windows Services 10
6. Testphase 12
6.1 Testdurchführung 12
7. Fazit 13

7.1 Soll-/ Ist-Vergleich 13
7.2 Lessons Learned 14
7.3 Ausblick für die Zukunft 14
A. Anhang ..... A
a. Glossar ..... A
b. Quellenverzeichnis ..... B
c. Detaillierte Zeitplanung ..... B
d. Unit Tests C
e. Gesamttest ..... D
f. ER-Diagramm ..... G
g. Relationales Datenmodell ..... H
h. Klassendiagramme I
i. Ausgewählter Source Code ..... K

Tabellenverzeichnis

Tabelle 1: Gantt-Diagramm für die Planung der Projektphasen 3
Tabelle 2: Zeitplan 4
Tabelle 3: Betriebsmittel 4
Tabelle 4: Gegenüberstellung Geplante Zeit/ Benötigte Zeit 14
Tabelle 5: Detaillierter Zeitplan ..... B

Abbildungsverzeichnis

Abbildung 1: Skizze des Windows Service 8
Abbildung 2: Unit Tests für das Konfigurationstool C
Abbildung 3: Eintrag einer Filterregel ..... D
Abbildung 4: Eintrag eines E-Mail Servers für Benachrichtigungen ..... D
Abbildung 5: Screenshot vom laufenden Service mit Task-Manager und Event Log ..... E
Abbildung 6: Eingang einer Benachrichtigung via E-Mail ..... E
Abbildung 7: Ausschnitt des Hauptformulars im Konfigurationstool ..... F
Abbildung 8: ER-Diagramm Logging Tool ..... G
Abbildung 9: Relationales Datenmodell ..... H
Abbildung 10: Klassendiagramm Konfigurationstool I
Abbildung 11: Klassendiagramm Logging Service ..... J
Abbildung 12: Methode zum Beziehen der Resultate (Konfigurationstool) ..... K
Abbildung 13: Methode IP-Adressenprüfung (Konfigurationstool) L

Hinweis:

Kursiv gesetzte Begriffe sind im Glossar erläutert.

1. Einleitung

1.1 Projektumfeld

Das Projekt wurde bei der ERP-Plus GmbH durchgeführt. ERP-Plus ist ein mittelständisches Unternehmen mit derzeitig acht Mitarbeitern und wird vom Dipl.-Math. J[redacted] als Geschäftsführer geleitet. Das Unternehmen ist in Hannover am Kröpcke ansässig und die Geschäftstätigkeit umfasst die Entwicklung und Pflege von Individual-Software sowie dazugehörige Support- und Beratungsleistungen.

1.2 Projektziel

Es soll ein Systemdienst für die Entwicklungsserver entworfen werden, der es ermöglicht, ein- und ausgehende *TCP/IP* und *UDP* - Datenübertragungen zu überwachen und zu protokollieren.

1.3 Projektbegründung

Vor einiger Zeit wurde von einem der verschiedenen Entwicklungsserver durch eine unbekannte Anwendung eine automatisch generierte E-Mail versandt, die daraufhin in einer *Spam Trap* gelandet ist. Spam Traps sind spezielle E-Mail-Adressen, die unter Beobachtung stehen und beispielsweise von Spamfilter-Herstellern zur Spambekämpfung eingesetzt werden. Sollten E-Mails von einem Server in einer Spam Trap landet, bedeutet dies in Konsequenz, dass der Server in seiner *Reputation* sinkt. Sinkt die Reputation unter einem bestimmten Schwellenwert, landet der entsprechende Server auf einer *Blacklist* und ankommende E-Mails werden automatisch geblockt.

Ein Reputationsverlust würde einen wirtschaftlichen Schaden nach sich ziehen. Dies soll verhindert werden und es gilt, die Reputation des Entwicklungsservers zu wahren.

1.4 Projektschnittstellen

Das Projekt wurde von Herrn J[redacted] als Geschäftsführer und Auftraggeber genehmigt. Die notwendigen Projektmittel wurden von der ERP-Plus GmbH zur Verfügung gestellt. Der Kreis der zukünftigen Anwender für dieses Projekt umfasst erfahrene Systemadministratoren.

1.5 Projektabgrenzung

Das Projekt war ein Einzelprojekt und somit kein Bestandteil eines größeren Gesamtprojektes. Im Zuge der Entwicklung des Dienstes wurden sowohl das Konfigurationstool als auch der Systemdienst vom Autor selbstständig programmiert.

2. Analyse

2.1 Ist-Zustand

Zum Zeitpunkt des Projektantrags lag im Unternehmen noch kein Dienst für Windows vor, um eine protokollierte Überwachung der TCP/IP und UDP - Datenübertragungen auf den Entwicklungsservern zu ermöglichen.

2.2 Soll-Zustand

Bei der Entwicklung der Individual-Software kommen bei ERP-Plus verschiedene Entwicklungsserver zum Einsatz. Bei einem dieser Server sollen die ein- und ausgehenden TCP-Verbindungen überwacht werden. Außerdem sollen auch die Ereignisse in Bezug auf UDP-Datenübertragungen erfasst werden.

Zusätzlich sollen dazu die dazugehörigen Prozesse mit Prozessnamen und Prozesspfad ermittelt werden. Zudem soll es möglich sein, Einsicht in die Menge der ein- und ausgehenden Datenübertragungen nehmen zu können.

Um diese Aufgabe zu erfüllen, sollte ein Dienst entworfen werden, der es erlaubt, Ereignisse von aufgebauten Verbindungen und erfolgten Datenübertragungen zu erfassen. Die erfassten Daten sollen anhand eigener Regeln gefiltert werden, um die gewonnenen Datenresultate für spätere statistische Auswertungen in einer relationalen Datenbank zu speichern. Zudem soll eine Art Alarmierung implementiert werden, die es ermöglicht, aufgrund erfasster Resultate E-Mails an zuvor ausgewählte Personen wie zum Beispiel einen Systemadministrator zu verschicken, um sie darüber in Kenntnis zu setzen, dass unerlaubte Datenübertragungen zustande gekommen sind.

Die Definition der Filterregeln soll über ein Konfigurationstool erfolgen, in dem die verschiedenen Filterregeln zur Einschränkung der Aufzeichnung aufgestellt werden können. Außerdem soll es möglich sein, dort die Firmen-E-Mail-Adressen der zu kontaktierenden Personen und entsprechend angepasste E-Mail-Texte zu hinterlegen. Die erfassten Parameter werden in eine Datenbank gespeichert, auf die auch der Dienst Zugriff hat.

Die Parameter für die Filterregeln sollen neben Angaben zu Remote-Adressen und Port des lokalen Servers auch Angaben zu Remote-Ports umfassen. Anhand dieser Filterregeln soll es auch möglich sein, ganze Bereiche abdecken zu können. Zudem soll eine Wahl zwischen den beiden Protokollarten TCP und UDP möglich sein.

Die Parameter, welche über das Konfigurationstool eingegeben werden, sollen in einer relationalen Datenbank gespeichert werden. Beim Start des Dienstes sollen die Parameter dann ausgelesen und der Dienst dementsprechend daraufhin konfiguriert werden.
Die notwendige Verbindungszeichenkette zur Datenbank soll bei der Installation in einer *Config-Datei* hinterlegt werden.

2.3 Wirtschaftlichkeitsanalyse

Als internes Einzelprojekt ergibt sich nur ein indirekter wirtschaftlicher Nutzen. Der Systemdienst erzielt an sich keine Umsatzsteigerung oder Kosteneinsparung. Deshalb wurde auf eine Amortisationsrechnung verzichtet. Dennoch ist die Wahrung der Reputation der Entwicklungsserver ein wichtiges Ziel, um auch in Zukunft den Prozess der Softwareentwicklung reibungslos zu gewähren. Landet die Domain oder die IP-Adresse eines Entwicklungsservers erst mal auf verschiedenen Blacklists, hat das negative wirtschaftliche Folgen, da dadurch jegliche E-Mails, welche von diesem Server ausgehen, zukünftig abgewiesen werden.

2.3.1 Make or Buy-Entscheidung

Auf dem Markt sind zwar Anwendungen vorhanden, die solch eine Überwachung ermöglichen. Das Betriebssystem Microsoft Windows selbst stellt an System-Tools den Taskmanager wie auch die Eventanzeige bereit. Dennoch erfüllt keins der verfügbaren Tools und Anwendungen die gestellten Anforderungen. Der Versand der reputationsschädigenden E-Mails ereignet sich nur sporadisch und eine längere, protokollierte Überwachung von vorher eingestellten Ports und IP-Adressen ist damit nicht möglich. Daher wurde die Entscheidung zugunsten einer Neuerstellung getroffen.

2.3.2 Projektkosten

Genaue Projektkosten konnten mangels valider Daten nicht beziffert werden. Es wurden nur Punkte angeführt, die bei einer Kostenaufstellung anfallen. Da keine Beschaffungen getätigt werden mussten, fielen auch keine Anschaffungskosten an. Was an notwendigen Ressourcen gebraucht wurde, war bereits vorhanden. Für die Nutzung des Rechners fielen entsprechende Stromkosten an.

Beim Einsatz der Ressourcen wurde auf bereits vorhandene Softwarelizenzen zurückgegriffen oder Open-Source Software eingesetzt, die eine kostenfreie, kommerzielle Nutzung miteinschließt.
Das verwendete Onlinetool draw.io schließt ebenso die kostenfreie, kommerzielle Nutzung ein. Somit fielen keine Lizenzkosten an. Zusammenfassen lassen sich diese Kosten als Gemeinkosten.
Ein Großteil der Projektkosten würde durch die anfallenden Personalkosten entstehen. Als Praktikant arbeitete der Autor unentgeltlich.

Dafür muss man die Zeit in Rechnung stellen, die für das Fachgespräch mit dem Projektleiter J[redacted] über den Projektauftrag und dem Sollzustand, wie auch durch die spätere Vorstellung der Lösungsansätze anfielen. Diese Kosten lassen sich über Einzelkosten abdecken.

2.4 Projektnutzen

Die Entwicklung und Einführung des datenbankgestützten Logging-Tools hilft dabei, potenziell schädliche Prozesse auf den Entwicklungsservern aufzudecken, um entsprechende Maßnahmen zur Wahrung der Server-Reputation einleiten zu können.

3. Planung

3.1 Projektphasen

				Do, 01 .Okt	Fr, 02 .Okt	Sa, 03 .Okt	So, 04 .Okt	Mo, 05 .Okt	Di, 06 .Okt	Mi, 07 .Okt	Do, 08 .Okt	Fr, 09 .Okt	Sa, 10 .Okt	So, 11 .Okt	Mo, 12 .Okt	Di, 13 .Okt	Mi, 14 .Okt	Do, 15 .Okt	Fr, 16 .Okt
Ablaufpunkt	**Startdatum**	**Enddatum**	**Dauer**																
Analyse	01.10.2020	01.10.2020	1 Tag																
Planung und Entwurf	02.10.2020	05.10.2020	2 Tage																
Umsetzung	06.10.2020	14.10.2020	7 Tage																
Dokumentation	15.10.2020	16.10.2020	2 Tage																

Tabelle 1: Gantt-Diagramm für die Planung der Projektphasen

2.3.3 Grobe Aufteilung der Phasen in Stunden

Phase	Geplante Zeit
Analyse	6 Stunden
Planung und Entwurf	10 Stunden
Umsetzung	34 Stunden
Test	8 Stunden
Dokumentation	12 Stunden

Tabelle 2: Zeitplan

3.2 Ressourcenplanung

Betriebsmittel	Beschreibung
Räumlichkeiten der ERP-Plus GmbH	
Schreibtisch mit Bürostuhl	
Ein Desktop-Rechner	Intel I-5 3330 3GHz mit 8 GB RAM, HDD, Maus und Tastatur
Zwei Monitore	Auflösung: Full-HD 1920-1080 Pixel
Ein Drucker	Laserdrucker/Scanner-Kombigerät
Integrierte Entwicklungsumgebung	Visual Studio 2019 Community Edition
Relationales Datenbankmanagementsystem	Microsoft SQL Server Express
Versionsverwaltungssoftware	Open Source Versionsverwaltung Subversion
Office Paket Software mit Nutzungslizenz	Microsoft Office Professional Plus 2016
Applikation zur Erstellung der Diagramme	Online-Anwendung Draw.io
Ein weiterer Desktop-Rechner	Zum Testen mit der Telnet-Konsole

Tabelle 3: Betriebsmittel

3.3 Vorgehensmodell für das Projekt

Bei der Auswahl eines geeigneten Vorgehensmodells schieden schon viele von vornherein aus. So ist zum Beispiel die agile Methode SCRUM lediglich bei Teamprojekten sinnvoll. Das Spiralmodell ist für dieses Projekt nicht angemessen, zumal dann auch immer das Zwischenprodukt vorgezeigt werden müsste, was sich negativ auf die beanspruchte Zeit auswirkt.

So wurde vom Autor das klassische Wasserfallmodell gewählt, weil das Projekt lediglich ein überschaubares Einzelprojekt war. Prinzipiell sind beim klassischen Wassermodell keine Rücksprünge vorgesehen. Die nächste Phase wird erst begonnen, wenn die vorige Phase abgeschlossen wurde.

4. Entwurf

4.1 Zielplattform für das Projekt

Die Zielplattform des Services ist ein Entwicklungsserver, auf dem Windows als Betriebssystem läuft und eine SQL-Datenbank von Microsoft zum Einsatz kommt.

Da im Praktikumsunternehmen hauptsächlich mit der objektorientierten Programmiersprache C# programmiert wird, wurde sie auch für die Implementierung des Projektes ausgewählt.
In Verbindung mit C# standen für die Erstellung der GUI für das Konfigurationstool Windows Forms und WPF zur Auswahl. *Windows Forms* ist schon älter, aber eignet sich gut, um relativ zügig GUIs zu erstellen. Dafür ist es schlechter skalierbar als *WPF*. Zudem bringt *WPF* eine stärkere Trennung von GUI und Programmcode mit sich.

Die Vorzüge von WPF waren aber für das Projekt irrelevant. Dagegen sprach auch der knappe Zeitrahmen des Projektes. Daher wurde für die Implementierung Windows Forms gewählt.

4.2 Datenschutz

Der Datenschutz ist bei einem Tool, dass unter anderem IP-Adressen aufzeichnet und mit E-Mail-Adressen arbeitet, sehr wichtig. Datenschutz umfasst die Regeln zur Erhebung, Verarbeitung und Speicherung sogenannter personenbezogener Daten.

Während der Planungsphase wurde festgelegt, dass das Zugangspasswort zum E-Mail-Server nicht offen in der Datenbank stehen darf, sondern entsprechend zu verschlüsseln ist. E-Mail-Adressen ebenso wie IP-Adressen zählen nach dem Bundesdatenschutzgesetz und der europäischen Datenschutz-Grundverordnung (DSGVO) zu den personenbezogenen Daten.

Personenbezogene Daten umfassen alle Daten, die eindeutig einer Person zugewiesen sind. So sehen die gesetzlichen Regelungen unter Anderem vor, dass es einer Einwilligung der Betroffenen bedarf, um diese Daten überhaupt zu verarbeiten und zu speichern. Gerade auch im Bezug auf die IP-Adressen ist das ein sensibles Thema, da es hierbei auch abzuwägen gilt, welches Interesse höher wiegt, also Datenschutz oder die Sicherheit. Für die E-Mail-Adressen sind nur reine Firmen-E-Mail-Adressen vorgesehen. Es wurde geplant, diese Felder ebenso zu verschlüsseln.

4.3 Entwurf des Entity Relationship Diagramms

Für den Entwurf des ER-Diagramms (siehe Abbildung 8, Anhang S. G) war es zunächst einmal wichtig herauszufinden, welche Entitäten prinzipiell beteiligt sind und wie die Kardinalitäten aussehen. Die erste Entität beschreibt die Datenübertragungen, die am Server ein- und ausgehen.

Die zweite Entität bilden die Filterregeln, die auf die Datenübertragungen angewendet werden. Dabei ergibt sich schon eine M:N-Beziehung, da sowohl viele Datenübertragungen auf viele Filterregeln treffen können als auch viele Filterregeln auf viele Datenübertragungen angewendet werden können.
Bei diesem Prozess sollen die Datenübertragungen geloggt werden, auf die eine der Filterregeln zutrifft. Der Rest wird verworfen. Als Folge daraus können sich auch viele Resultate bilden, welche in die Datenbank zur späteren Auswertung geschrieben werden.

4.4 Entwurf des relationalen Datenmodells

Beim Entwurf (siehe Abbildung 9, Anhang S. H) wurde für die Entitäten Filterregel und Resultat jeweils eine Tabelle aufgestellt. Namentlich wurden sie als FilterRule und DataTransmissionResult gekennzeichnet. Die Attribute für beide Entitäten leiten sich aus den Anforderungen ab. So umfasst das Resultat neben den Angaben zur Remoteadresse, den Portangaben und den dahinterstehenden lokalen Prozessen inklusive Prozessname und Pfadangabe auch eine Angabe zur Protokollart und einen Verweis auf die entsprechende Filterregel.

Um diesen Verweis bewerkstelligen zu können, war es notwendig, eine Hilfstabelle zwischen dem Resultat und der Filterregel einzuführen. So löste sich die M:N-Beziehung in zwei 1:N-Beziehungen auf.

4.5 Entwurf der GUI für das Konfigurationstool

Für die Erstellung der Mockups wurde die IDE Visual Studio genutzt, da es mit Hilfe von Windows Forms leicht möglich ist, entsprechende Mockups zu erstellen, ohne ein externes Tool dafür zu beanspruchen. Das Konfigurationstool spaltet sich in vier Formulare auf.
Das Hauptformular stellt eine Ansicht der gefundenen Resultate an Datenübertragungen dar. Darüber hinaus soll es von dort auch möglich sein, die Daten in der Tabelle DataTransmissionResult wieder zu löschen.

Das zweite Formular zeigt eine Übersicht über alle aufgestellten Filterregeln. Dort soll es auch möglich sein, entsprechend neue Filterregeln anzulegen, zu löschen oder zu aktualisieren. Dazu wurden entsprechende Buttons angelegt. Ein Button ermöglicht es auch, sämtliche Regeln zu löschen. Die Voraussetzung dafür soll aber sein, dass zuvor auch sämtliche Resultate gelöscht wurden. Ansonsten könnte man nicht mehr herausfinden, nach welcher Filterregel ein Resultat geloggt wurde und es würde eine sogenannte Löschanomalie in der Datenbank auftreten.

Das dritte Formular ist direkt mit dem zweiten Formular assoziiert. Es soll sich öffnen, wenn der Administrator eine neue Regel erstellen möchte oder eine bestehende Regel verändern möchte. Hierüber werden auch die Benachrichtigungstexte erfasst, die später beim E-Mail-Versand an zuvor definierte Mitarbeiter auftauchen sollen.

Das letzte Formular erfasst die Zugangsdaten zum E-Mail-Server, um später die Benachrichtigungsmails verschicken zu können.

4.6 Entwurf der Programmlogik für das Konfigurationstool

Die Programmlogik wurde zusammen mit den Formularen in einem Klassendiagramm (siehe Abbildung 8, Anhang S. I) aufgezeichnet. Neben den eigentlichen Eventmethoden, die sich hinter den einzelnen Buttons auf den Formularen befinden, wurden Hilfsmethoden eingeplant, um die Eingaben auf Plausibilität prüfen zu können.

Daneben wurde eine Klasse namens Encryption eingeplant, die dafür sorgen soll, dass das Zugangspasswort für den E-Mail-Server nicht offen, sondern verschlüsselt in der Datenbank steht.

Neben den drei Datenklassen für die E-Mail-Serverkonfiguration, die Filterregeln und die Resultate aus den Abfragen des Services wurde noch eine Klasse DBTableCRUD eingeplant, die im Kontakt mit der Datenbank steht und entsprechende Methoden bereitstellt, um die notwendigen SQL-Abfragen ausführen zu können.

4.7 Entwurf des Services

Um den Service entwerfen zu können, musste erstmal ermittelt werden, wie die Daten zu beschaffen sind. Um die notwendigen Daten abfragen zu können, wurde vom Autor die Internet Protocol Helper API, oder auch kurz IP Helper herausgesucht. Diese API gehört zum Betriebssystem Windows und ermöglicht als Bestandteil der Win32 API, Netzwerkdaten von der lokalen Maschine abzufragen.
Diese API basiert auf den Programmiersprachen C bzw. C++, da auch das Betriebssystem Windows zum Teil damit programmiert wurde. Allerdings gibt es bereits bei Pinvoke.net, einem Wikipedia für .Net Entwickler ein Gerüst, um diese Daten auch für C#-Anwendungen und Dienste bereitzustellen.

Die Herausforderung beim Übergang von C bzw. C++ und C# besteht unter Anderem in den unterschiedlich verwendeten Datentypen. Zudem handelt es sich bei C und C++ aus Sicht von C# um sogenannten Unmanaged Code. C# als objektorientierte Sprache besitzt die Möglichkeit, Objekte freizugeben, die keinerlei Referenz mehr aufweisen.
Damit müssen sich Entwickler beim Implementieren weniger um das Speichermanagement kümmern. Die Voraussetzung für diesen Mechanismus ist allerdings, dass es sich dabei um Managed Code handelt.

Die Funktionen, auf die über die API zugegriffen wird, lauten GetExtendedTcpTable und GetExtendedUdpTable. Damit lässt sich schon ein Großteil der notwendigen Daten abfragen.
Um die restlichen, fehlenden Daten abfragen zu können, wurde damit geplant, die Methodenbibliothek von .Net-Framework zu nutzen.

Allerdings ließen sich damit nicht die Datenraten der ein- und ausgehenden Datenübertragungen abfragen. Dafür bedarf es des Zugriffs auf das Event Tracing für Windows, kurz ETW. Wie schon der Name erahnen lässt, können damit sogenannte Events abgefragt werden. In Verbindung für einzelne Prozesse lassen sich damit auch Datenraten abfragen. Allerdings zeigte sich schon zu dem Zeitpunkt aus der Erfahrung mit einer kleinen Testanwendung, dass die Abfrage bzw. die spätere Implementierung sich, wenn überhaupt, nur sehr schwierig gestalten lassen würde. Zum einen wäre der Einsatz einer Listener-Methode notwendig gewesen. Mit dieser Methode ließen sich einzelne Prozesse abhorchen. Aber damit viele Prozesse gleichzeitig zu überwachen, gestaltet sich sehr schwierig. Der knappe Zeitrahmen für das Projekt ließ auch keine tiefere Recherche mehr zu und so wurde für diese Anforderung nur eine geringe Priorität eingeräumt.

Im Zuge der weiteren Planung wurde der Service als Multithreading-Service (siehe Abbildung 2, Seite 9) entworfen. Ein Singlethread-Service wäre für die gestellten Anforderungen nicht ausreichend gewesen. Es gibt vier Bereiche, in die sich der Service aufteilt. Ein Thread wurde dafür geplant, die Daten von den oben erwähnten Funktionen über die API abzufragen.

Vier Threads wurden dafür geplant, die empfangenen Daten anhand der Filterregeln zu durchsuchen und gefundene Ergebnisse zwischenzuspeichern.

Ein Thread war dafür angedacht, Kontakt zur Datenbank aufzunehmen, um Filterregeln abzufragen und Resultate aus dem vorigen Filterprozess in der Datenbank zu speichern. Außerdem sollten darüber auch gegebenenfalls die E-Mails versendet werden. Und der Hauptthread soll dazu dienen, die anderen Threads zu starten und zu überwachen und bei Fehlern den Service zu stoppen.

Um den Datenaustausch zwischen den einzelnen Threads zu gewährleisten wurden Klassen eingeplant, die als gemeinsam genutzte Stapel bzw. Warteschlangen, sogenannte Queues, dienen sollen.
Dafür gibt es in C# threadsichere Auflistungen, die nach dem *FIFO-Prinzip* funktionieren und zudem ermöglichen, dass unterschiedliche Threads darauf zugreifen können, ohne dass es zu Konflikten kommt. Dafür werden intern Synchronisierungsmechanismen eingesetzt, die Fehler vermeiden sollen, wenn zwei oder mehr Threads auf eine gemeinsam genutzte Queue zugreifen.

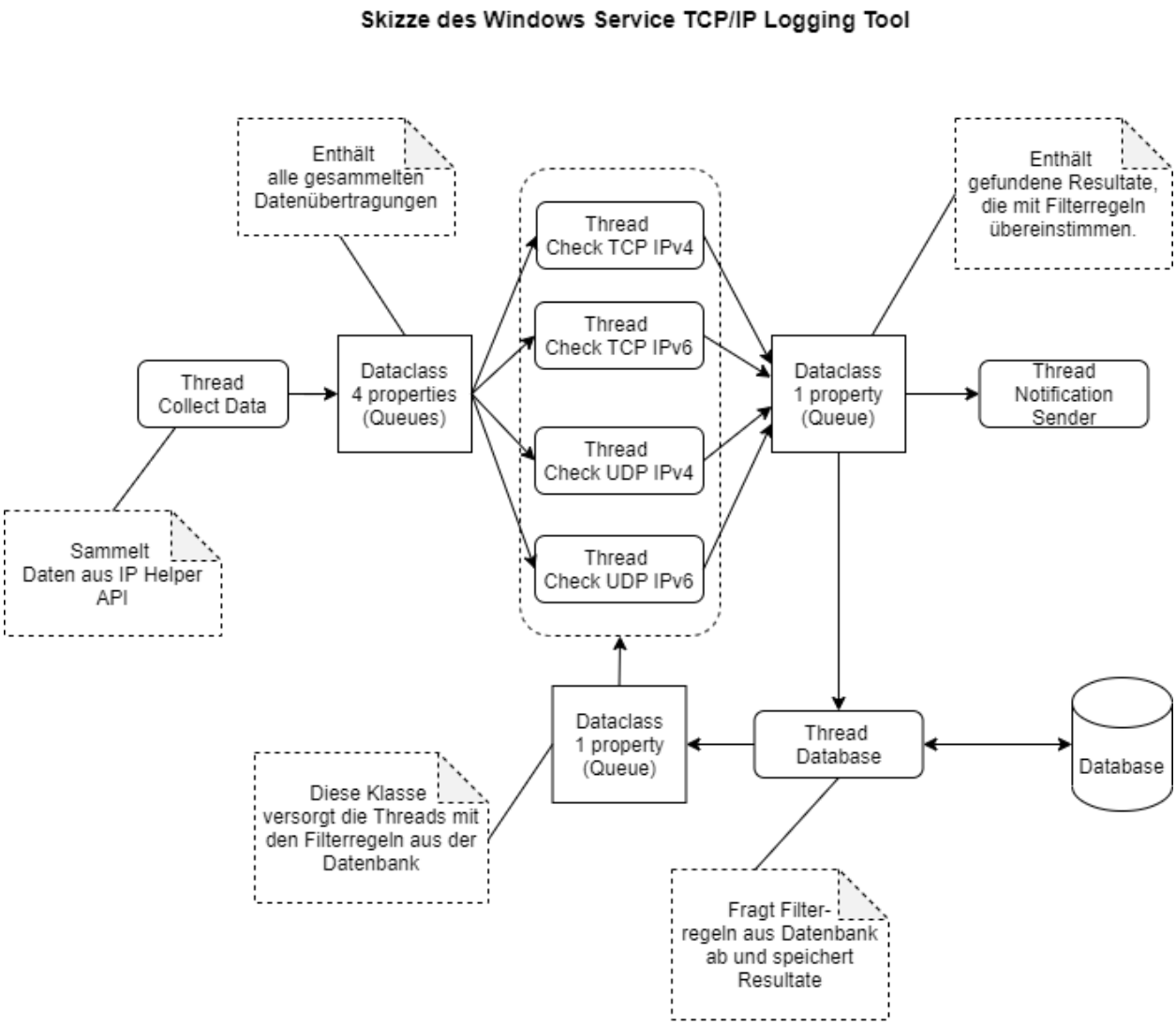

Abbildung 1: Skizze des Windows Service

5. Implementierung

5.1 Versionsverwaltung

Zur Versionsverwaltung wurde Subversion mit der grafischen Erweiterung Tortoise für Windows verwendet. Subversion ist eine zentrale Versionsverwaltung, die sich von dezentralen Versionsverwaltungen wie zum Beispiel Git dadurch abgrenzt, dass es nur ein zentrales Repository gibt, in denen alle Versionen und Versionsverläufe verwaltet werden.
Der Autor hatte sich trotz eigenem Git-Account für Subversion entschieden, da er allein am Projekt gearbeitet hat. Zudem wird sie auch im Praktikumsunternehmen verwendet.

5.2 Implementierung der Datenbank

Die erste Implementierung der Datenbank wurde auf dem Arbeitsrechner des Autors vorgenommen. Visual Studio bietet dazu die Möglichkeit, sich im Rahmen der Entwicklung eine Instanz des Microsoft SQL-Server anzulegen. Dieses Feature wird als SQL Server Express bezeichnet und ermöglicht eine relativ schnelle und unkomplizierte Art eigene Anwendungen mit Datenbankanbindung zu testen.

Zuerst wurden die Tabellen anhand des entworfenen Datenbankmodells händisch implementiert und mit Attributen versehen. Danach wurden die Tabellen entsprechend verknüpft. Bei der Erstellung werden von Visual Studio automatisch Skripte generiert, die auf Transact-SQL basieren. Transact-SQL ist eine Erweiterung von SQL und ist eng mit der Funktionalität von Microsoft SQL Servern verflochten.
Die automatisch erstellten Skripte wurden später im Konfigurationstool implementiert, um eine Generierung der notwendigen Tabellen und Attributen zu ermöglichen.

5.3 Implementierung des Konfigurationstools

Ausgangsbasis für die Implementierung des Konfigurationstools waren die Mockups, die nun mit Programmlogik hinterlegt werden mussten. Zuerst wurde damit begonnen, die zwei Formulare für die Einstellung der Filterregeln zu implementieren.

Für die Übersicht der Filterregeln wurde ein schlichtes Datagridview eingebunden. In ihr sind alle vorhandenen Regeln mit den wichtigsten Parametern aufgelistet. Ein Update einer dieser Regeln öffnet ein neues Formular, wo die übernommenen Parameter aus der Übersicht wiederauftauchen. Das gleiche Formular wird auch geöffnet, wenn man eine neue Regel anlegt.

Die Implementierung selbst gestaltete sich anfangs als unproblematisch. Aber schon bei der Prüfung der eingegebenen IP-Adressen stellte sich die Frage, wie sichergestellt werden kann, dass es sich bei den Eingaben um gültige Adressen handelt. Da auch zusätzlich eine Filterung auf eine zuvor definierte Range, also ein bestimmter Adressbereich möglich sein soll, muss sichergestellt werden, dass die Adressen bei der Eingabe nicht vertauscht wurden.

Für die Prüfung der IP-Adresse wurde ein regulärer Ausdruck, kurz Regex, eingesetzt. Da der Entwurf eines solchen Ausdrucks viel Zeit in Anspruch nehmen kann, suchte der Autor nach bereits vorhandenen Ausdrücken, die sowohl für IPv4 als auch IPv6 Adressen funktionieren. Für das Problem der vertauschten Eingabe der Adressen wurde eine Methode implementiert, welche die Adressen in Byteblöcke zerlegt und Byte für Byte miteinander vergleicht.

Auch für die Eingabefelder bezüglich Lokal- und Remote-Port wurden Prüfmethoden angelegt. Die Eingabefelder, die für die Erstellung von Benachrichtigungs-E-Mails zuständig sind, sind optional und es erfolgt dort keine weitere Prüfung auf Plausibilität. Nur die E-Mail-Adresse unterliegt einer Prüfung durch einen regulären Ausdruck.

Um einer möglichen SQL-Injektion entgegenzuwirken, bietet C# die Möglichkeit, die Eingaben zu parametrisieren. Dadurch stehen mögliche Kommandos, um zum Beispiel eine Drop-Anweisung auf eine Tabelle durchzuführen, als harmloser Eintrag im jeweiligen Datenfeld. Im Zuge der Parametrisierung wurden darüber auch die Datentypen festgelegt, mit der die Werte in der Datenbank erwartet wurden. Da der Autor keinen *objektrelationalen Mapper* wie Entity Framework als Brücke zwischen Anwendung und Datenbank verwendet hat, muss gerade dort darauf geachtet werden, dass die Datentypen passen. Der Autor hatte bei der Implementierung dahingehend ein paar Mal mit Ausnahmefehlern zu kämpfen, die erst zur Laufzeit auftraten.

Nach den Formularen für die Filterregeln wurde das Formular für die E-Mail-Server-Konfiguration mit Programmlogik erweitert. Auch hier unterliegen die Eingabefelder für E-Mail und den einzustellenden SMTP-Port jeweils einer Prüfmethode.

Parallel dazu wurde auch die Klasse Encryption zur Verschlüsselung angelegt, damit das Zugangspasswort nicht offen in der Datenbank auftaucht. Im sehr viel späteren Verlauf der Implementierung wurde die Verschlüsselung auf sämtliche E-Mail-Adressen und den Usernamen für den Serverzugang erweitert. Dem Autor wurde bewusst, dass es gerade in Bezug auf die E-Mail-Adressen datenschutzrechtliche Probleme geben kann, da diese zu den personenbezogenen Daten gehören und damit nicht offen in der Datenbank sein sollten.

Als letzter Punkt wurde beim Hauptformular noch ein *Datagridview* implementiert, welches beim Aufruf des Konfigurationstools die gewonnenen Daten anzeigt. Nach dem späteren abschließenden Test wurde beim dafür zuständigen Select-Kommando noch eine Limitierung eingeführt, da der Aufruf des Konfigurationstools sonst zu lange dauert, da erst die ganzen Datensätze zur Ansicht ausgelesen werden.

5.4 Implementierung des Logging-Tools bzw. Windows Services

Ein Windows Service läuft im Gegensatz zu einer Konsolenanwendung im Hintergrund und zeigt keinerlei direkte Ausgaben. Dies erschwert aber das Debugging, weshalb der geplante Service zuerst als Konsolenanwendung erstellt wurde und erst kurz vor Ende der Implementierungsphase zu einem Service mit spezifischen Methoden zum Starten und Beenden umgewandelt wurde.

Bei der Implementierung wurde zuerst mit dem Zugang zur IP Helper API begonnen. Die Vorlage dafür ist schon bei Pinvoke.net vorhanden und wurde hierbei mit einigen Anpassungen übernommen. Die damit verbundenen Methoden und Strukturen unterteilen sich in einen Abschnitt für TCP und einen Abschnitt für UDP. Zusätzlich wird zwischen IPv4 und IPv6 unterschieden. Dies ermöglicht eine gezielte Abfrage nach Protokoll und IP-Adressformat.

Die bereits vorhandenen Strukturen wurden vom Autor erweitert, um zusätzliche Daten wie zum Beispiel Prozess bzw. Prozessname, Prozesspfad oder die Zeit zu erfassen.

Um herauszufinden, welche Prozesspfade und welcher Art die Verbindung ist, also ob ein- oder ausgehend, wurden zusätzliche Methoden implementiert. Allerdings zeigten sich dabei schon Einschränkungen, die auch bis zum Projektende nicht behoben werden konnten. Bei Systemprozessen werden keine Prozesspfade angezeigt. Die damit verbundene ManagementObjectSearcher Klasse, die zur Abfrage verwendet wurde, stellt diese Informationen nicht zur Verfügung.

Viel gravierender ist allerdings die Einschränkung, dass nur bei TCP-Verbindungen angezeigt werden kann, ob es sich dabei um eine ein- oder ausgehende Datenübertragung handelt.
Die verwendete Funktion GetExtendedUDPTable zur Abfrage nach UDP-Datenübertragungen stellt keine Struktur bereit, die Rückschlüsse auf die Remote-Adresse oder Remote-Port zulässt. Zudem handelt es sich bei UDP um ein verbindungsloses Protokoll. Der Autor hatte bei der Implementierung keine praktische Lösung mehr gefunden, die Daten den UPD- bzw. IP- Headern zu entlocken.

Nachdem die Anbindung an die API implementiert wurde, folgte die Erstellung einer Datenklasse DataTransmissionQueue, die vier Warteschlangen bzw. Queues zur Aufnahme der Listen beinhaltet. Die gewonnenen Daten aus der API kommen als stark typisierte Listen an und werden aufgeteilt. Zwei Listen sind für TCP zuständig und zwei für UDP.

An diesen vier Warteschlangen bedienen sich die vier implementierten Threads ThreadDataCheckTCPv4, ThreadDataCheckTCPv6, ThreadDataCheckUDPv4 und ThreadDataCheckUDPv6, welche die Daten mit den definierten Filterregeln vergleichen und gegebenenfalls als DataTransmissionResult in eine weitere, gemeinsam genutzte Warteschlange geben. Zum Zeitpunkt der Erstellung wurden die dafür zuständigen Methoden aber noch offengelassen. Stattdessen kümmerte sich der Autor erstmal um die Implementierung der Klassen und Methoden, die direkt in Verbindung mit der Datenbank stehen. Hierbei traten kaum Probleme auf, da der Autor aus den Erfahrungen bei der Erstellung des Konfigurationstools gelernt hatte.

Nachdem die Anbindung an die Datenbank abgeschlossen wurde, kümmerte sich der Autor um die Ausarbeitung der Methoden, die dafür sorgen sollen, dass die Daten anhand der Regeln ausgefiltert werden. Wie bereits angedeutet, werden diese Methoden von den vier Threads aufgerufen und laufen dann zur Laufzeit endlos weiter.

Bereits dabei musste sich der Autor schon um die Abstimmung der Threads kümmern. Die Abfragen und die zahlreichen Daten, die dabei anfallen, stellen für das System eine starke Belastung an Systemressourcen dar, wenn die dafür vorgesehenen Threads ungebremst laufen. Um die Last zu senken, wurden die Threads mit einem Sleep-Mechanismus belegt, um sie für einen gewissen Zeitraum schlafenzulegen. Das Timing wurde erstmal auf 2 Sekunden festgelegt, was bedeutet, dass die Threads zum Beschaffen der Daten und Vergleichen immer wieder regelmäßig für 2 Sekunden schlafen.

Die Speicherung der Resultate wird von einem weiteren Thread namens ThreadSaveResultData übernommen. Die Schwierigkeit dabei war, dass die Resultate aus vier Threads wieder in eine gemeinsame Warteschlange zusammengeführt werden und der Autor dabei die Befürchtung hatte, dass der Thread, der für die Speicherung in die Datenbank zuständig ist, nicht genug Zeit bekommen würde, die Warteschlange abzuarbeiten, ohne das der sogenannte Heap-Speicher, der die erstellten Objekte vorhält, langfristig überläuft. Aus diesem Grund wurde das Timing für den Sleep-Mechanismus bei dem betreffenden Thread auf 0,2 Sekunden abgesenkt.

Im Anschluss daran wurde entgegen der ursprünglichen Planung ein weiterer Thread namens ThreadSendMails dafür eingerichtet, der sich darum kümmert, dass die entsprechenden E-Mail-Benachrichtigungen versandt werden, die zusammen mit den Filterregeln zuvor definiert wurden. Der zusätzliche Thread wird damit begründet, dass die Kontaktaufnahme zum E-Mail-Server auch etwas Zeit benötigt. Damit dieser Thread nicht die anderen Threads bei ihren Aufgaben ausbremst, wurde eine letzte Warteschlange eingerichtet, die lediglich die Benachrichtigungstexte und E-Mail-Adressen zur Laufzeit vorhält. Auch hierbei wurde das Timing herabgesetzt, um genug Zeit zur Erfüllung der Aufgabe einzuräumen.

Zusätzlich wurde noch eine Begrenzung innerhalb des E-Mailversands implementiert, die verhindern soll, dass innerhalb von kurzer Zeit stets die gleiche Benachrichtigung an einen Empfänger versandt wird. Das Zeitfenster zwischen den Versand an den gleichen Empfänger wurde hierbei willkürlich auf 30 Minuten gestellt.

Zu guter Letzt wurde die erstellte Konsolenanwendung als Grundlage genommen, um daraus einen Windows Service zu erstellen. Hierbei ergaben sich erst Schwierigkeiten, da der Autor noch nicht mit dem genauen Ablauf eines Windows Service vertraut war und sich erst wunderte, warum der Service trotz erfolgreicher Einbindung und Start nicht anlief, wie erwartet. Das Problem war, dass zwar die OnStart Methode als Einstiegspunkt des Service aufgerufen wurde, aber darin dann weiterlief, ohne weitere Befehle abzuarbeiten. Der Autor hatte das Problem gelöst, in dem der Programmcode aus der Main-Methode in die OnStart Methode implementiert wurde.

6. Testphase

6.1 Testdurchführung

Die Testphase fiel deutlich kürzer aus als ursprünglich geplant. Das Testen des Konfigurationstools bestand aus einer Kombination von händischen Testeingaben und Durchführung von einigen Unit Tests für entsprechend zugängliche Methoden (siehe Abbildung 2, Anhang S. C und S. D Gesamttest).
Die GUI-Eventmethoden konnten aufgrund ihres Zugriffsmodifikators private nicht mit den Unit Tests abgedeckt werden und wurden händisch getestet. Dabei wurden unter Anderem bewusste Fehleingaben gemacht und der Versuch einer SQL Injection durchgeführt.

Für die Unit Tests wurde das Framework MS Test von Microsoft verwendet. Ein Code Coverage Test konnte nicht durchgeführt werden, da dieses Feature nur der Enterprise Version von Visual Studio zur Verfügung steht.

Die Tests für den Windows Service fanden zuerst an der Konsolenanwendung statt. Diese Tests bestanden aus manuellen Komponententests, welche an neuralgischen Stellen Ausgaben zur Überprüfung zu erzeugen. Zusätzlich wurde mit Hilfe eines zweiten Rechners und der Telnet Konsole ein Test durchgeführt, um zu sehen, ob eine eingehende TCP-Verbindung als solches auch später in der Auswertung angezeigt wird. Im Anschluss daran wurde ein Laufzeittest von über einer Stunde durchgeführt. Dabei wurde das Windows Event Log auf Fehlereinträge geprüft und anhand der privaten E-Mail des Autors die korrekte Funktionalität des E-Mail-Versands geprüft.

Während der ganzen Laufzeit ist kein Fehler aufgetreten und die E-Mails wurden wie erwartet, korrekt versandt. Aufgetretene Bugs während der Testphase wurden behoben, aber ein Codereview und eine Abnahme wurden nicht mehr durchgeführt. Die Zeit reichte dafür nicht mehr aus, um beides vernünftig zu absolvieren.

7. Fazit

7.1 Soll-/ Ist-Vergleich

Der entworfene Dienst erfüllt nicht alle gestellten Anforderungen. Die gestellte Anforderung, die Datenraten der ein- und ausgehenden Datenübertragungen für die einzelnen Prozesse zu überwachen, ist mit der verwendeten Windows API so nicht möglich.

Eine konstante Überwachung dieser Art erfordert Zugang zur Windows Event Tracing API. In Angesicht des knappen Zeitrahmens blieb nicht mehr ausreichend Zeit, um sich damit ausreichend vertraut zu machen und es zusätzlich zu implementieren.
Zudem unterscheidet sich die Windows Event Tracing im Ansatz von der verwendeten Windows API im Service. Es wäre notwendig gewesen, dass der Service ständig lauscht und nicht in bestimmten Zeitzyklen immer wieder eine Momentaufnahme erstellt.

Mit den erstellten Momentaufnahmen tritt ein weiteres Problem auf. Je kürzer die Zeitabstände zwischen den erstellten Momentaufnahmen sind, desto mehr Systemressourcen benötigt der Dienst. Im schlimmsten Fall ist das System völlig ausgelastet, was nicht im Sinne eines Windows Dienstes sein kann. So musste der Autor künstliche Zeitabstände implementieren, was zur Folge hat, dass eine lückenlose Überwachung nicht möglich ist.

Eine weitere Problematik stellt die Genauigkeit der Resultate dar. Es war aus den Anforderungen nicht ersichtlich, ob eine sich wiederholende Datenübertragung nach einer bestimmten Zeit ignoriert werden oder weiter aufgezeichnet werden sollte. Wenn zum Beispiel eine längere andauernde TCP-Verbindung entsteht, auf die eine der Filterregeln zutrifft, so hat das zur Folge, dass innerhalb von kurzer Zeit viele Datenbankeinträge entstehen.

Des Weiteren ließ sich die angestrebte Forderung im Bereich UDP-Übertragungen nicht angemessen umsetzen, da die vom Autor ausgewählte Windows API die entsprechende Funktionalität nur unzureichend zur Verfügung stellt. Das hat zur Folge, dass eine Filterung auf Remote-Adressen oder Remote-Ports nicht möglich ist.

Die Testphase war wegen Zeitmangel unzureichend und ein Codereview steht aus. Insgesamt lässt sich feststellen, dass das Projekt, gemessen an den Anforderungen, nur teilweise erfolgreich war.

Phase	Geplante Zeit	Benötigte Zeit
Analyse	6 Stunden	6 Stunden
Planung und Entwurf	10 Stunden	12 Stunden
Implementierung	34 Stunden	40 Stunden
Test	8 Stunden	2 Stunden
Dokumentation	12 Stunden	10 Stunden
Summe	70 Stunden	70 Stunden

Tabelle 4: Gegenüberstellung Geplante Zeit/ Benötigte Zeit

7.2 Lessons Learned

Der Autor hat einiges über Threads und den verwendeten Warteschlangen als C#-Datentypen gelernt. Zudem wurde einiges über die Verwendung einer Windows API für die eingesetzten Funktionen gelernt.

In puncto Projektplanung und Umsetzung wurde ersichtlich, dass es relativ umfangreich war und einzelne Schritte mehr Zeit benötigten als ursprünglich eingeräumt wurden. Dieser Umstand schlug sich dann in einer unzureichenden Umsetzung einzelner Anforderungen beziehungsweise einer zu kurzen Testphase nieder.

7.3 Ausblick für die Zukunft

Ob der entworfene Service in der Form zum Einsatz kommen wird, bleibt unklar. Für die Verschlüsselung der personenbezogenen Daten muss eine bessere Lösung gefunden werden, denn bisher ist es zu leicht möglich, an den notwendigen Key heranzukommen, um die Daten wieder zu entschlüsseln.

Insgesamt bleibt auch der Datenschutz bei diesem Projekt ein offenes Thema. Die Frage, wie man mit der beabsichtigten Speicherung und Verarbeitung von IP-Adressen umgehen darf, ließ sich vom Autor nicht zweifelsfrei beantworten. Wäre das Projekt eine Webanwendung, so wäre auf jeden Fall ein Hinweis auf eine sogenannte Datenschutzerklärung zu setzen. So etwas ist bei einem Windows Service nicht möglich.

Bevor der Service überhaupt zum Einsatz kommen kann, muss erst mit dem Datenschutzbeauftragten des Betriebes beziehungsweise mit dem Datenschutzbeauftragten des Landes geklärt werden, wie lange die Daten überhaupt gespeichert werden dürfen. Zusätzlich kann man über eine Implementierung zur Verschleierung der Daten nachdenken. Außerdem sind die Mitarbeiter über den Einsatz des Service zu informieren.

Bei der Datenbankanbindung wäre zu überlegen, ob man das Konfigurationstool für ein breites Angebot an Servern implementiert. In der derzeitigen Form ist es spezifisch nur auf Microsoft SQL Server abgestimmt. Dazu wäre es auch überlegenswert, doch einen objektrelationalen Mapper wie Entity Framework oder Dapper einzusetzen oder zumindest auszuprobieren, wie es sich leistungstechnisch verhält.

A. Anhang

a. Glossar

Begriff	Beschreibung
Blacklist	Eine Liste, die aus Einträgen von IP-Adressen und Domaineinträgen mit schlechter Reputation besteht und dafür sorgt, dass versandte E-Mails von diesen Absendern automatisch abgewiesen werden.
Config-Datei	Diese Datei enthält Konfigurationsparameter, die zur Laufzeit der Anwendung abgefragt werden können. Bei diesem Projekt enthalten die Config-Dateien die Verbindungszeichenfolge für die Datenbank.
Datagridview	Datagridview ist ein Windows Forms Element zur Darstellung der Daten innerhalb einer tabellenartigen Struktur.
FIFO-Prinzip	FIFO bedeutet First In, First Out und steht im Zusammenhang mit der Datenstruktur Queue, die unter C# eine Auflistung darstellt.
Objektrelationaler Mapper	Die Mapper dienen als „Brücke“ zwischen den Datentypen der Datenbank und den Datentypen des Programmcodes.
Reputation / Sender Reputation	Dieser Wert, häufig zwischen 0 und 100, zeigt an, wie es um die Reputation eines Absenders steht. Fällt ein Absender häufiger durch Spam Mails auf, fällt dessen Reputationswert ab.
Spam Trap	Darunter versteht man speziell präparierte, nicht offensichtlich zugängliche E-Mail-Adressen, die dazu dienen sollen, unerwünschte Spam Mails aufzufangen und den Absender anhand der IP-Adresse zu identifizieren.
TCP	TCP bedeutet Transmission Control Protocol und befindet sich auf der ISO-OSI Schicht 4. TCP ist ein verbindungsorientiertes Protokoll, welches dazu dient, die Datenintegrität übertragender Datenpakete sicherzustellen.
TCP/IP	TCP/IP stellt eine Protokollfamilie dar, die sich aus den Protokollen IP, TCP und UDP zusammensetzt.
UDP	UDP bedeutet User Datagram Protocol und umschreibt ein verbindungsloses Protokoll, welches auf der 4. Schicht im ISO-OSI Modell angeordnet ist. Im Gegensatz zu TCP wird die Datenintegrität nicht sichergestellt.
Windows Forms	Windows Forms ist eine Bibliothek für .Net zum Erstellen von GUI-Applikationen.
WPF	WPF bedeutet Windows Presentation Foundation und ist ebenso eine reichhaltige Bibliothek zum Erstellen von GUI-Applikationen.

b. Quellenverzeichnis

Pinvoke.Net für Internet Protocol Helper API

(Zuletzt besucht am 16.10.2020)

https://www.pinvoke.net/default.aspx/iphlpapi/GetExtendedTcpTable.html

Microsoft.com für Internet Protocol Helper API, Event Tracing für Windows und Verschlüsselung

(Zuletzt besucht am 16.10.2020)

https://docs.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getextendedtcptable

https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing

https://docs.microsoft.com/de-de/dotnet/api/system.security.cryptography.aes?view=netcore-3.1

Digitalfortress.tech für Reguläre Ausdrücke

(Zuletzt besucht am 16.10.2020)

https://digitalfortress.tech/tricks/top-15-commonly-used-regex/

c. Detaillierte Zeitplanung

Phase und Beschreibung	Geplante Zeit
Analyse	
Fachgespräch mit dem Projektleiter über Projektauftrag und Sollzustand	2 Stunden
Aufnahme der Ist-Situation und des Projektumfeldes	1 Stunde
Recherche nach Alternativen und Abwägung zwischen Make und Buy	1 Stunde
Vorstellung der Lösungsansätze beim Projektleiter	2 Stunden
Planung und Entwurf	
Aufstellen eines Zeitplans und Festlegung auf Vorgehen	1 Stunde
Datenbankentwurf erstellen	1 Stunde
Mockups für Konfigurationstool erstellen	4 Stunden
Erstellung der Diagramme	4 Stunden
Umsetzung	
Einrichtung der Datenbank und der dafür notwendigen Tabellen	2 Stunden
Implementierung des Konfigurationstools auf Basis der Mockups	12 Stunden
Implementierung der API	2 Stunden
Implementierung der Klassen und Methoden zur Verarbeitung der Daten	8 Stunden
Erstellung der Klassen zur Anbindung an die Datenbank	6 Stunden
Implementierung der Funktion zum Versenden von Benachrichtigungen	4 Stunden
Test	
Testen des Konfigurationstools und Unittests	6 Stunden
Gesamttest	2 Stunden
Dokumentation	
Erstellung der Dokumentation	12 Stunden
Summe	**70 Stunden**

Tabelle 5: Detaillierter Zeitplan

d. Unit Tests

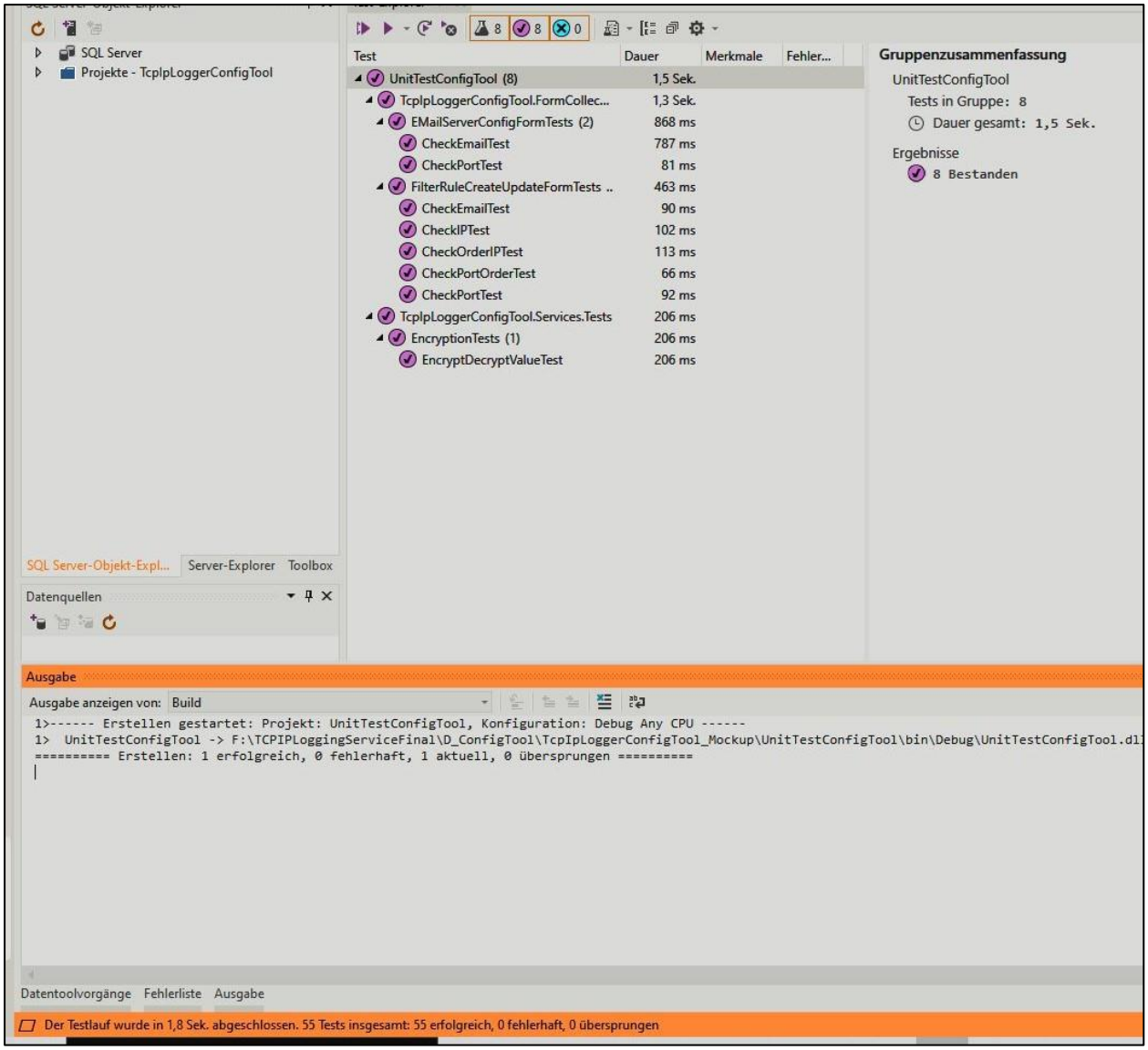

Abbildung 2: Unit Tests für das Konfigurationstool

e. Gesamttest

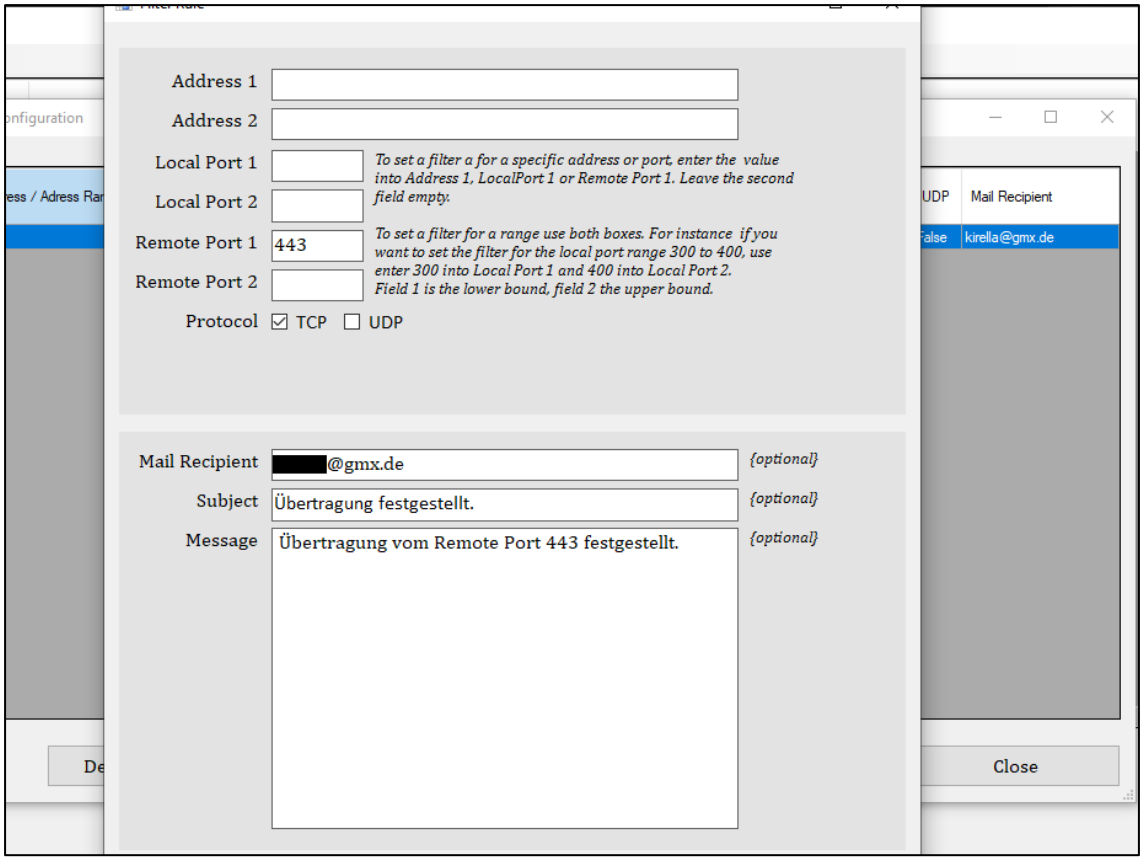

Abbildung 3: Eintrag einer Filterregel

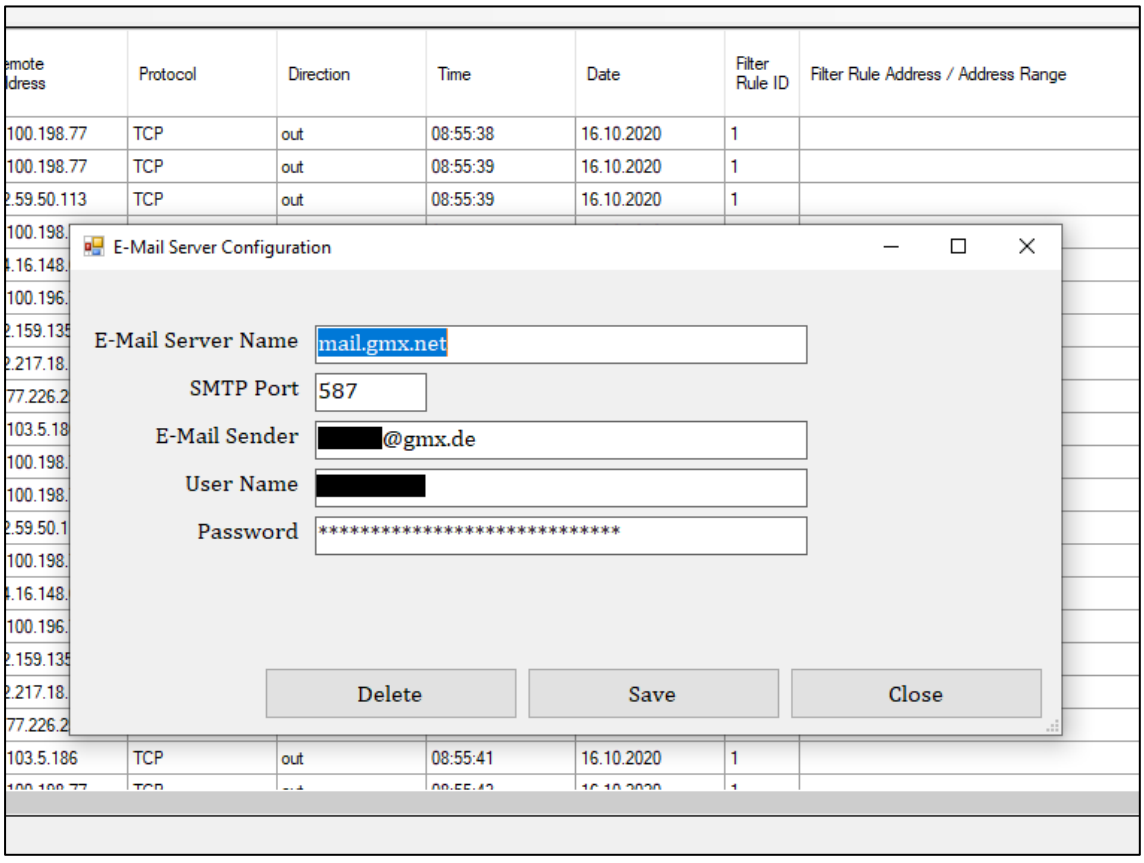

Abbildung 4: Eintrag eines E-Mail Servers für Benachrichtigungen

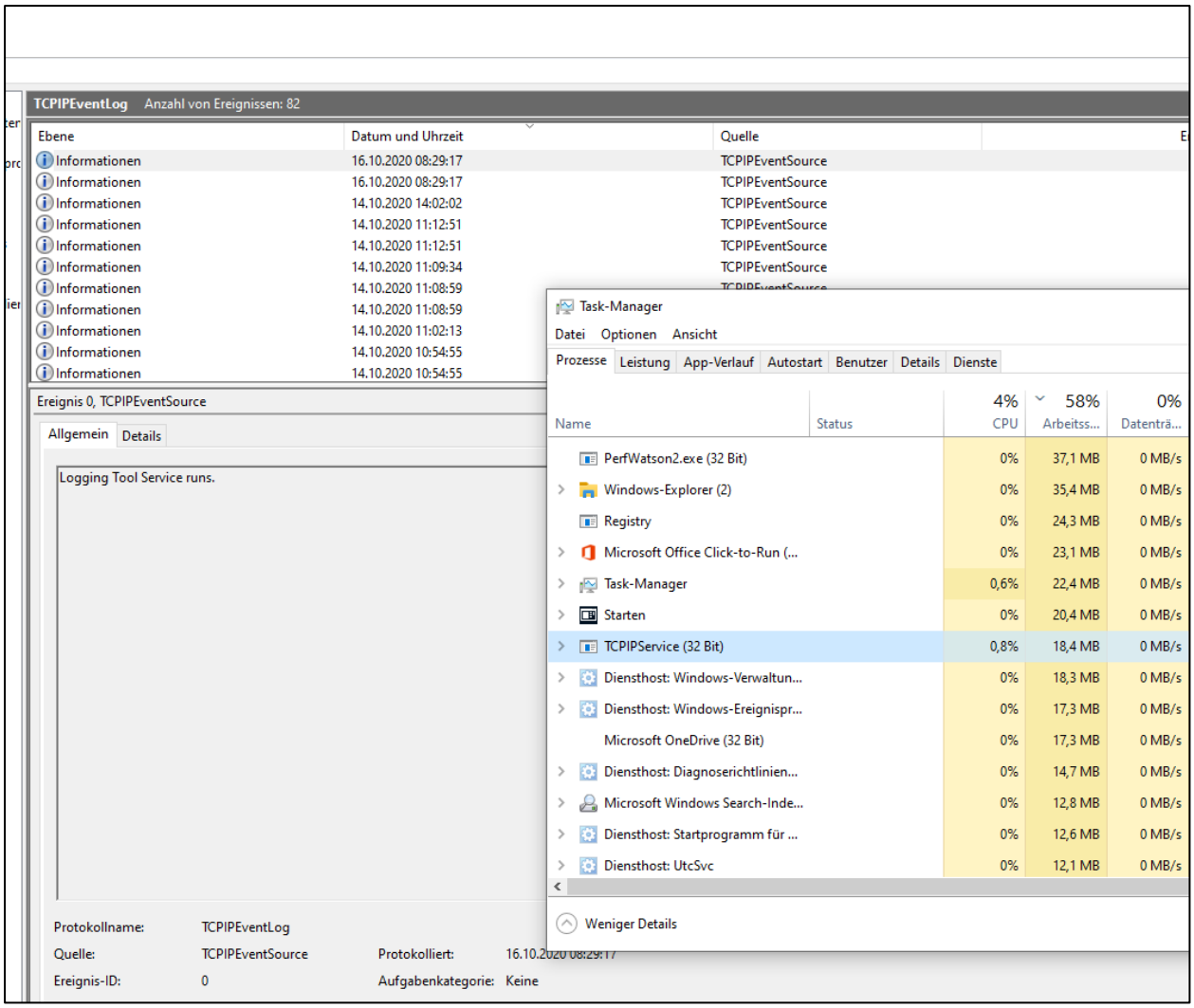

Abbildung 5: Screenshot vom laufenden Service mit Task-Manager und Event Log

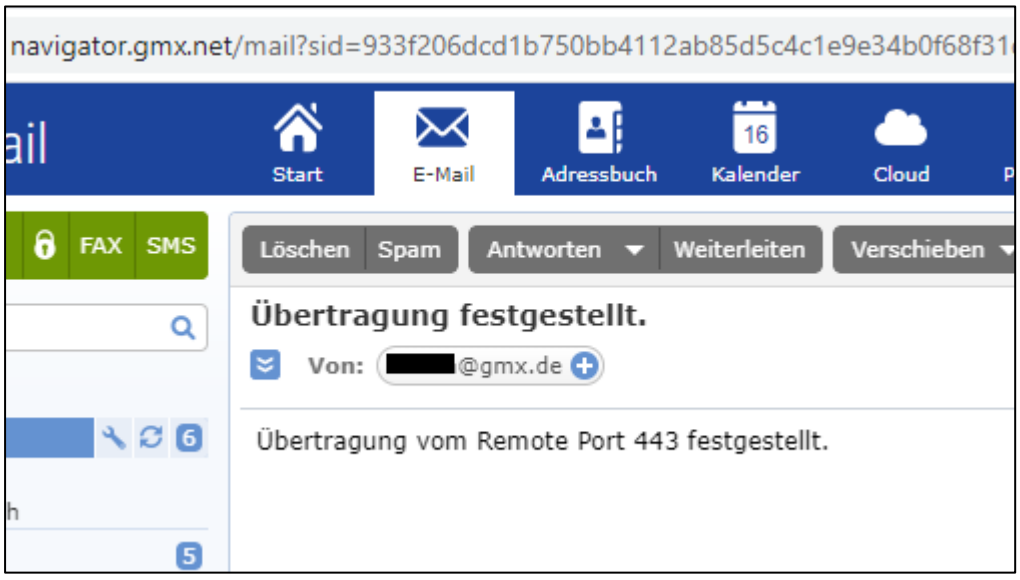

Abbildung 6: Eingang einer Benachrichtigung via E-Mail

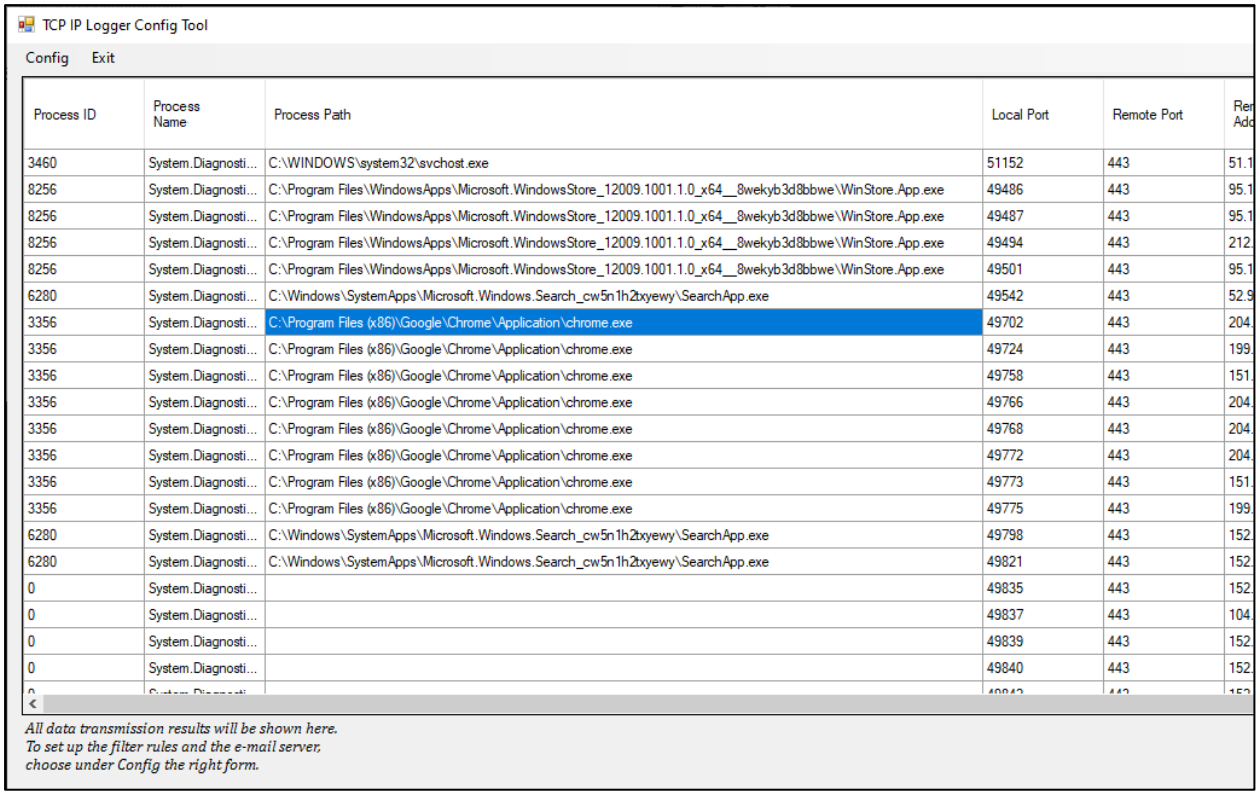

Abbildung 7: Ausschnitt des Hauptformulars im Konfigurationstool

f. ER-Diagramm

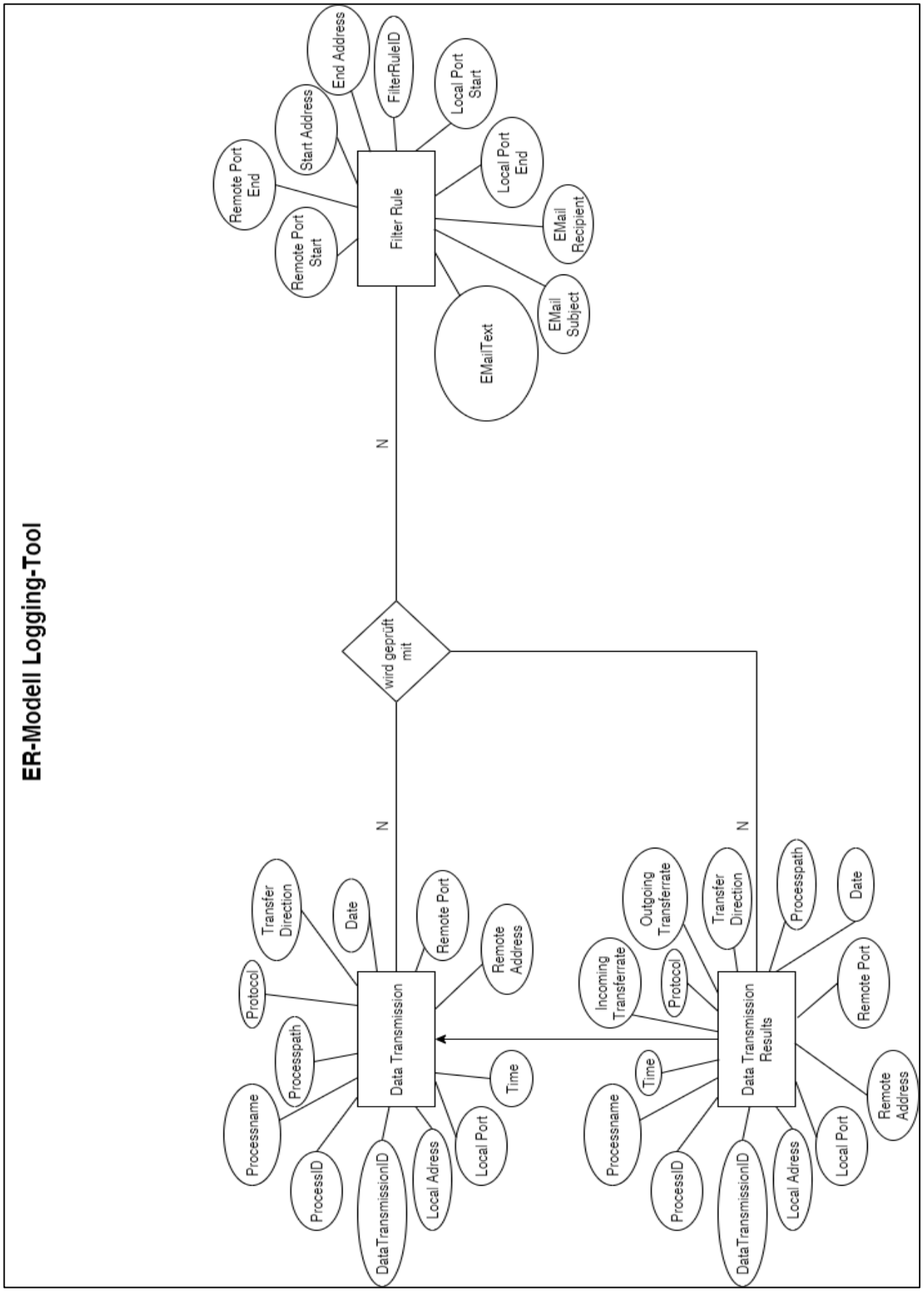

Abbildung 8: ER-Diagramm Logging Tool

g. Relationales Datenmodell

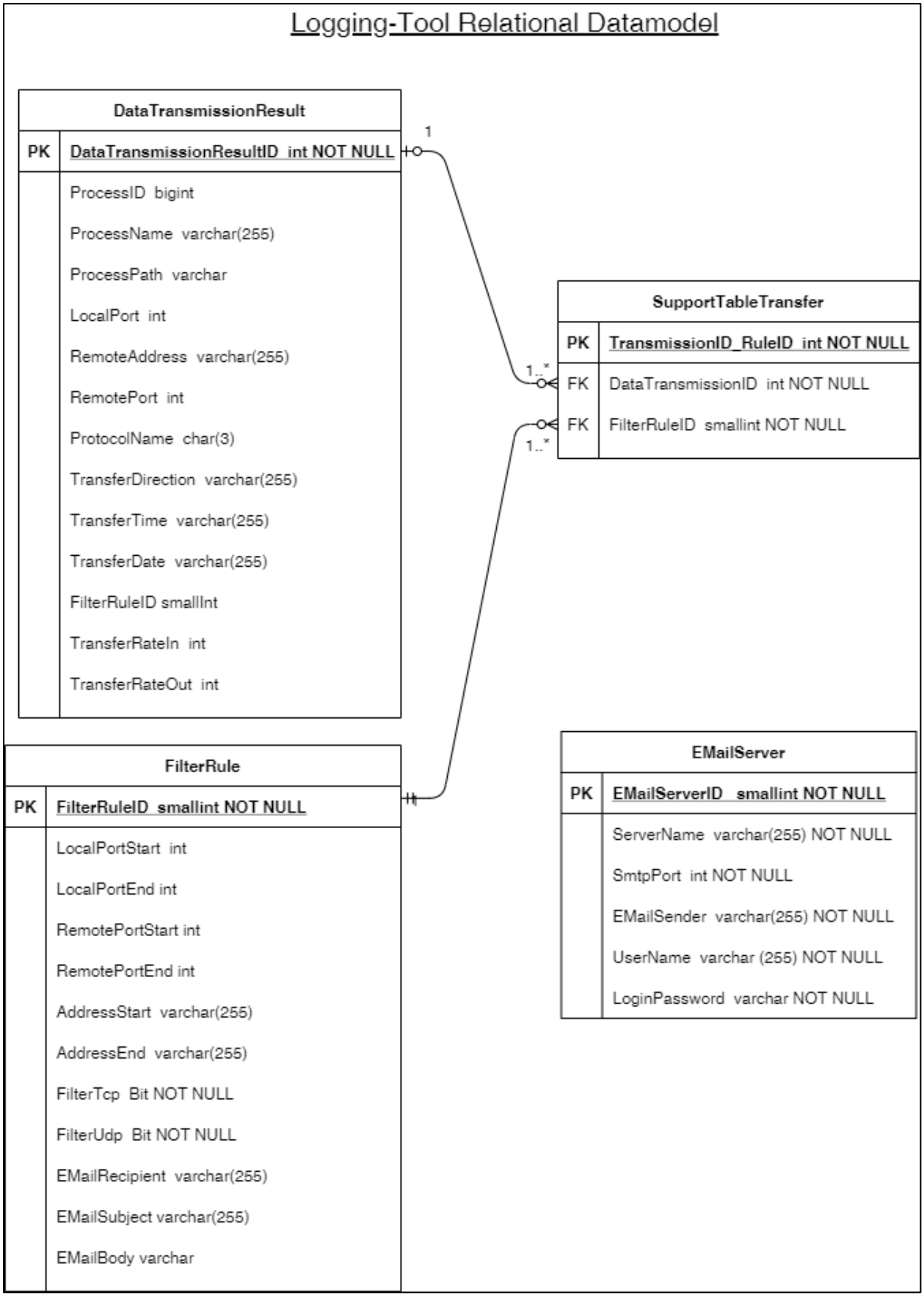

Abbildung 9: Relationales Datenmodell

h. Klassendiagramme

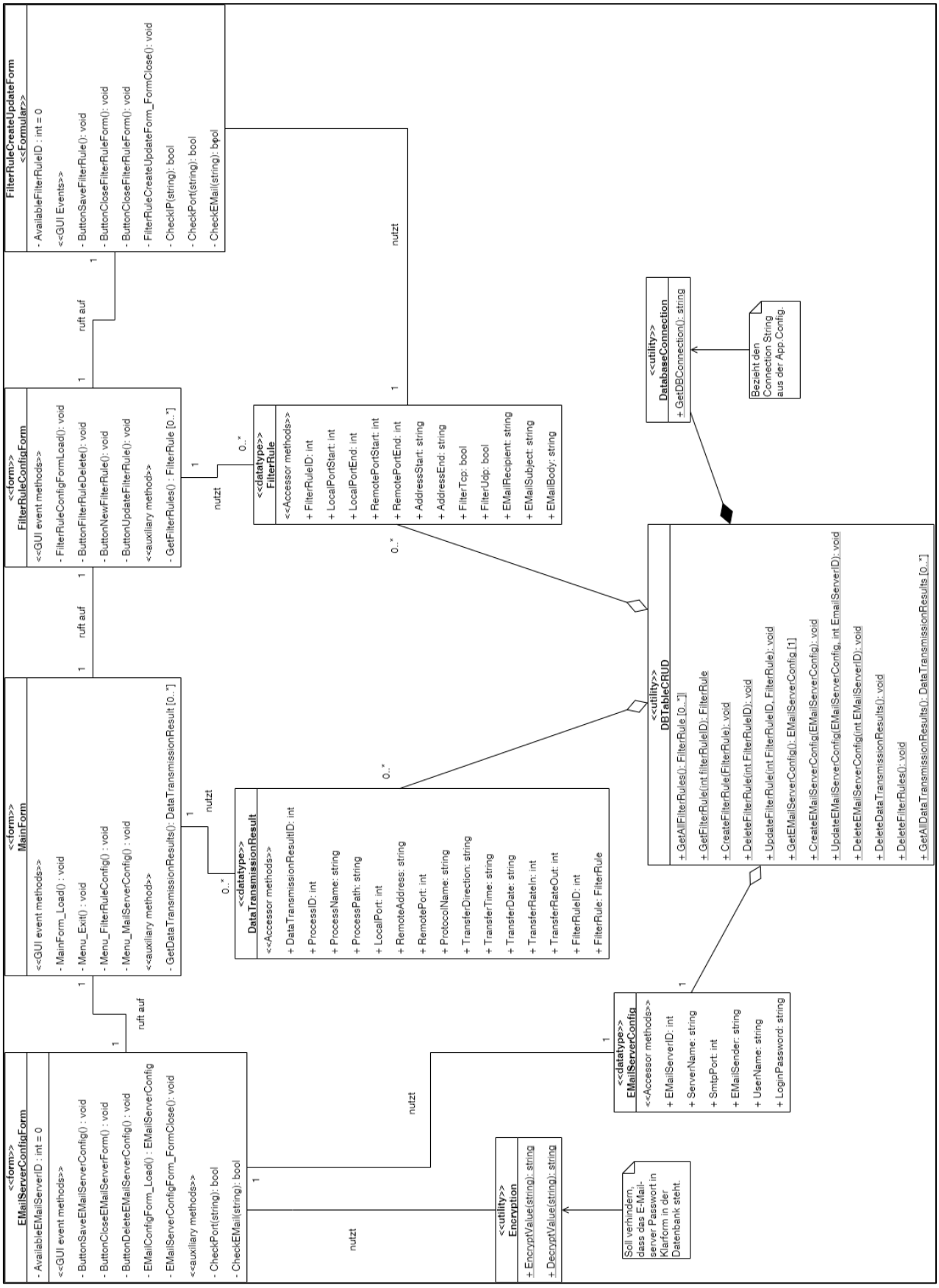

Abbildung 10: Klassendiagramm Konfigurationstool

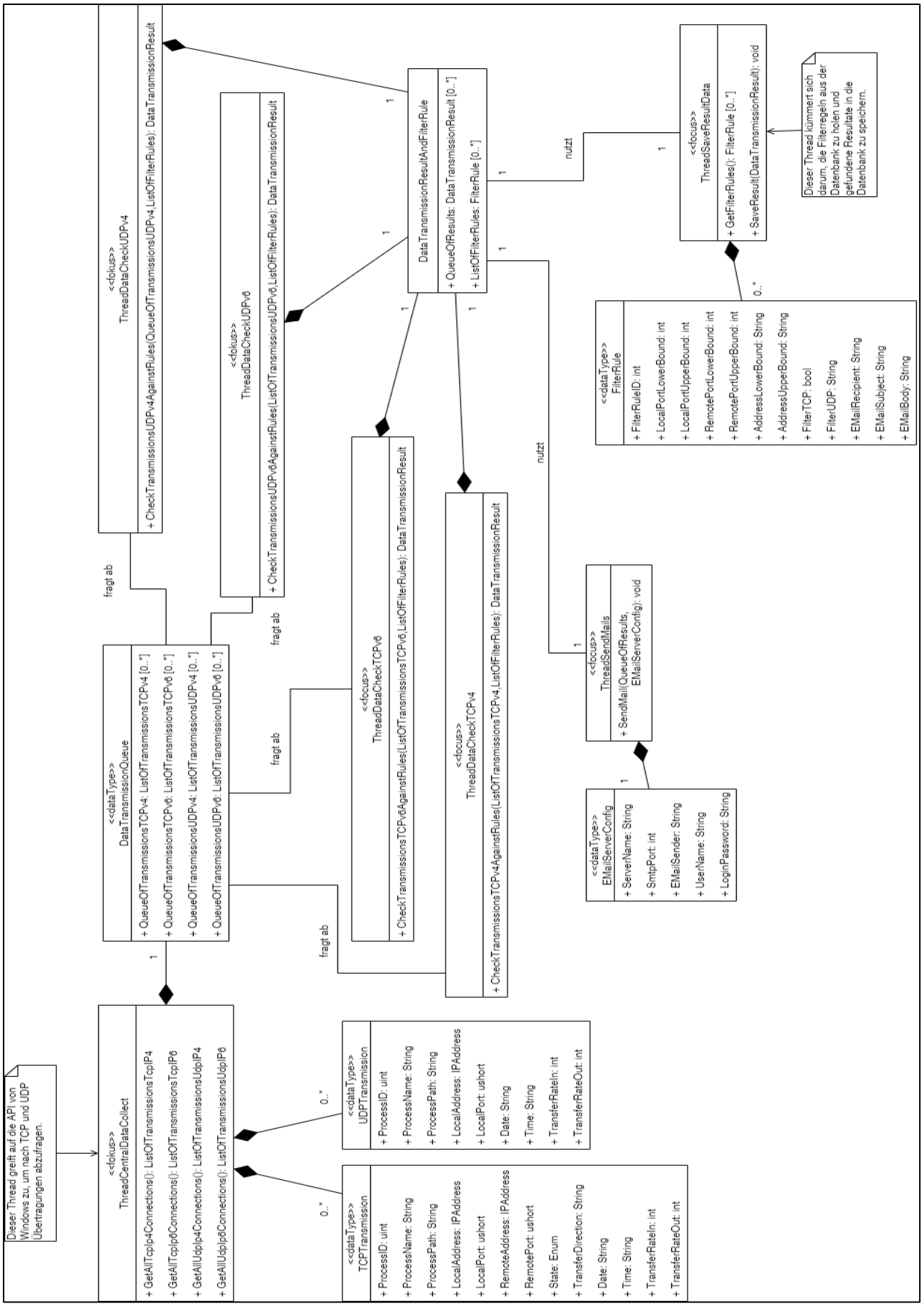

Abbildung 11: Klassendiagramm Logging Service

i. Ausgewählter Source Code

```
1-Verweis
public static bool GetAllDataTransmissionResults(out List<DataTransmissionResult> listOfResults)
{
    try
    {
        listOfResults = new List<DataTransmissionResult>();
        string connString = DatabaseConnection.GetDBConnection();
        string SQLCmd = "SELECT TOP 100 * FROM DataTransmissionResult " +
                        "JOIN SupportTableTransfer ON " +
                        "DataTransmissionResult.DataTransmissionResultID = SupportTableTransfer.DataTransmissionResultID " +
                        "JOIN FilterRule ON " +
                        "SupportTableTransfer.FilterRuleID = FilterRule.FilterRuleID;";

        using (SqlConnection connection = new SqlConnection(connString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(SQLCmd, connection);
            SqlDataReader reader = command.ExecuteReader();
            while (reader.Read())
            {
                DataTransmissionResult result = new DataTransmissionResult();
                if (reader.HasRows)
                {
                    if (!reader.IsDBNull(0)) result.DataTransmissionResultID = reader.GetInt32(0);
                    if (!reader.IsDBNull(1)) result.ProcessID = reader.GetInt64(1);
                    if (!reader.IsDBNull(2)) result.ProcessName = reader.GetString(2);
                    if (!reader.IsDBNull(3)) result.ProcessPath = reader.GetString(3);
                    if (!reader.IsDBNull(4)) result.LocalPort = reader.GetInt32(4);
                    if (!reader.IsDBNull(5)) result.RemoteAddress = reader.GetString(5);
                    if (!reader.IsDBNull(6)) result.RemotePort = reader.GetInt32(6);
                    if (!reader.IsDBNull(7)) result.ProtocolName = reader.GetString(7);
                    if (!reader.IsDBNull(8)) result.TransferDirection = reader.GetString(8);
                    if (!reader.IsDBNull(9)) result.TransferTime = reader.GetString(9);
                    if (!reader.IsDBNull(10)) result.TransferDate = reader.GetString(10);
                    if (!reader.IsDBNull(11)) result.FilterRuleID = reader.GetInt16(11);
                    if (!reader.IsDBNull(12)) result.TransferRateIn = reader.GetInt32(12);
                    if (!reader.IsDBNull(13)) result.TransferRateOut = reader.GetInt32(13);
                    if (!reader.IsDBNull(18)) result.LocalPortStart = reader.GetInt32(18);
                    if (!reader.IsDBNull(19)) result.LocalPortEnd = reader.GetInt32(19);
                    if (!reader.IsDBNull(20)) result.RemotePortStart = reader.GetInt32(20);
                    if (!reader.IsDBNull(21)) result.RemotePortEnd = reader.GetInt32(21);
                    if (!reader.IsDBNull(22)) result.AddressStart = reader.GetString(22);
                    if (!reader.IsDBNull(23)) result.AddressEnd = reader.GetString(23);
                    if (!reader.IsDBNull(24)) result.FilterTcp = reader.GetBoolean(24);
                    if (!reader.IsDBNull(25)) result.FilterUdp = reader.GetBoolean(25);
                    if (!reader.IsDBNull(26)) result.EMailRecipient = reader.GetString(26);
                    if (!reader.IsDBNull(27)) result.EMailSubject = reader.GetString(27);
                    if (!reader.IsDBNull(28)) result.EMailBody = reader.GetString(28);
                }
                if (!string.IsNullOrEmpty(result.EMailRecipient))
                {
                    result.EMailRecipient = Encryption.DecryptString(result.EMailRecipient);
                }
                listOfResults.Add(result);
            }
        }
        return true;
    }
    catch
    {
        listOfResults = new List<DataTransmissionResult>();
e Probleme gefunden
```

Abbildung 12: Methode zum Beziehen der Resultate (Konfigurationstool)

```
5 Verweise | 1/1 bestanden
public bool CheckIP(string IPAddressEntry)
{
    string pattern = @"((^\s*((([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.)" +
                     @"{3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]))\s*$)|" +
                     @"(^\s*((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|(([0-9A-Fa-f]{1,4}:)" +
                     @"{6}(:[0-9A-Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|" +
                     @"1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|" +
                     @":((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|" +
                     @"(([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25[0-5]|" +
                     @"2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|" +
                     @"(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:" +
                     @"((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|" +
                     @"(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]{1,4}){0,3}:" +
                     @"((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|" +
                     @"(([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:((25[0-5]|" +
                     @"2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(:(((:[0-9A-Fa-f]" +
                     @"{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|" +
                     @"2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:)))(%.+)?\s*$))";

    bool validIPAddress;
    if (Regex.IsMatch(IPAddressEntry, pattern) == false)
    {
        validIPAddress = false;
    }
    else
    {
        validIPAddress = IPAddress.TryParse(IPAddressEntry, out IPAddress ConvertedIPAddress);

        List<IPAddress> invalidIPAddresses = new List<IPAddress>
        {
            IPAddress.Parse("0.0.0.0"),
            IPAddress.Parse("0.0.0.1"),
            IPAddress.Parse("255.255.255.255"),
            IPAddress.Parse("127.0.0.1")
        };

        foreach (IPAddress item in invalidIPAddresses)
        {
            if (item.Equals(ConvertedIPAddress))
            {
                validIPAddress = false;
                break;
            }
        }
    }
    return validIPAddress;
}
```

Abbildung 13: Methode IP-Adressenprüfung (Konfigurationstool)