sniffdatel

SW Engineering Projekt FS 2016

Software Architektur

David Meister, Giorgio Vincenti, Samuel Krieg, Andreas Stalder

26. April 2016

1 Änderungsgeschichte

Datum	Version	Änderung	Autor
15.03.16	0.9	Erstellung erster Version	Giorgio Vincenti
18.04.16	1.0	Review ready	alle
26.04.16	1.1	Korrektur	alle

Tabelle 1: **Änderungsgeschichte**

Inhaltsverzeichnis

2 Einführung

2.1 Zweck

In diesem Dokument wird die Architektur von **sniffdatel** erläutert. Es soll als Wegweiser dienen und Hilfestellung für die Entwicklung bieten.

2.2 Gültigkeitsbereich

Das Dokument bezieht sich auf die Software **sniffdatel**, welche im Rahmen des Moduls Engineeringprojekt des Frühjahrssemester 2016 entwickelt wird und ist während des gesamten Projekts gültig.

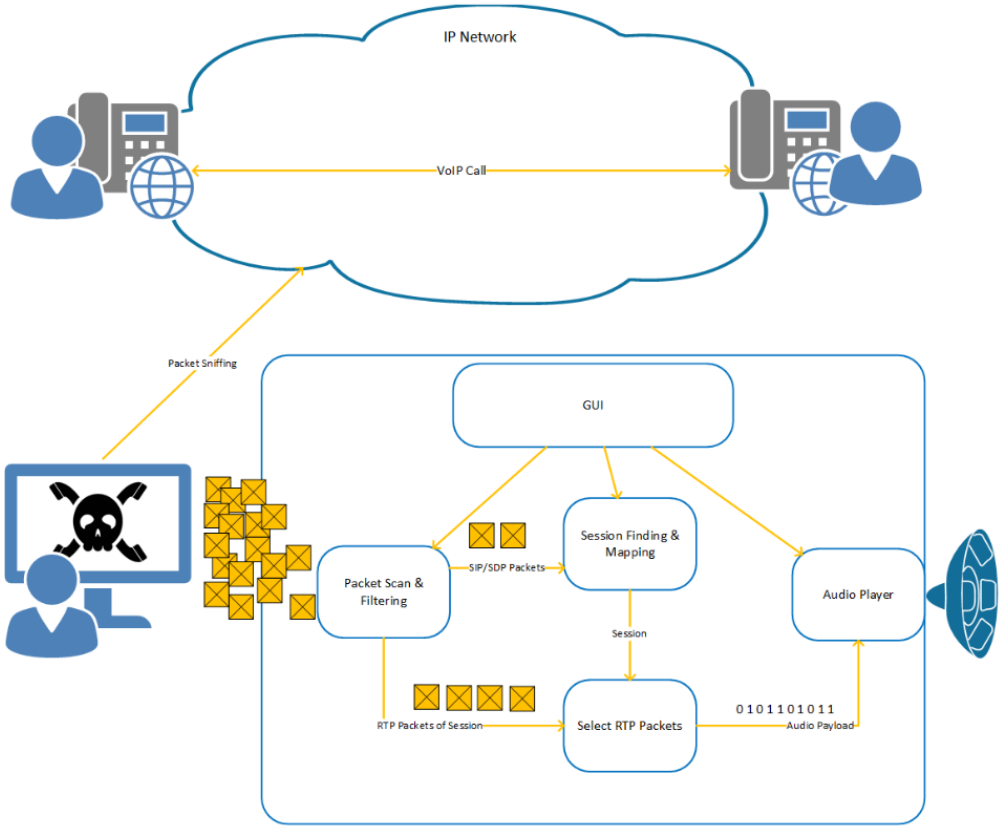
2.3 Referenzen

- doc/02_projektplan/Projektplan_v1.2
- doc/03_analyse/Anforderungsspezifikationen_v1.0
- doc/03_analyse/Domainanalyse_v1.0
- doc/04_architektur/externes_design_v1.0

Diese Dokumente sind bei uns auf der Dropbox oder auf Github zu finden. Für den Betreuer sind die Dokumente auf Redmine vorhanden.

3 Systemübersicht

In der nachfolgenden Abbildung ist eine Darstellung des Systems ersichtlich. Die Applikation lässt sich auf einer beliebigen Linux Distribution mit installiertem JRE (Java Runtime Environment) ab Version 8 starten. Über das GUI wird der Packet Scanner gestartet. Der Packet Scanner wir mit der jnet pcap - und das GUI mit der JavaFX Library realisiert. Die Applikation hat eine eigene Filterlogik implementiert um Sessions zu finden, anzuzeigen und die dazugehörigen RTP Pakete weiter zu verarbeiten. Die RTP Payloads werden in einen Buffer geschrieben und der Audioplayer (javax.sound Library) spielt den RTP Payload ab.



4 Architektonische Ziele und Einschränkungen

4.1 Ziele

- Die Software soll benutzerfreundlich gestaltet werden. Es sollte jedem möglich sein, ohne grosses Know-How, ein VOIP Gespräch im Netzwerk abzuhören.
- Die Software sollte von einem beliebigen Linux Client gestartet und genutzt werden können.
- Die Benutzeroberfläche der Software soll einfach und übersichtlich gestaltet sein.
- Die Software soll in jedem beliebigen Netzwerk funktionieren.

4.2 Einschränkungen

- Die Software muss auf den Clients als Administrator ausgeführt werden, damit der Zugriff auf die Netzwerkschnittstellen sauber funktioniert.
- Die Software kann keine Verschlüsselte Pakete wiedergeben.
- Das Abspielen eines Gesprächs benötigt eine gewisse Zeit um die gesammelten RTP Pakete zwischenzuspeichern und abzuspielen. Es wird mit einer Verzögerung von maximal einer Sekunde gerechnet.
- Die Software kann nur neu eingehende VoIP Anrufe erkennen und als Session im GUI anzeigen. Laufende VoIP Gespräche werden ignoriert.
- Es kann nur der G.711(µ-law/A-law) Codec abgespielt werden.
- Die Software beschränkt sich auf UDP-Verkehr und erkennt somit Gespräche welche TCP involvieren nicht.
- Die Software wird nur unter Linux Distributionen unterstützt.

4.3 Tools

astah: Werkzeugtool für das modellieren von Diagrammen während dem gesamten Projekt.

Eclipse: Entwicklungsumgebung für die Software.

Java 8: Es wird mit Java 8 entwickelt.

Texmaker: Alle Dokumente rund ums Projekt werden mit Texmaker erstellt.

Jnetpcap: Library um auf Netzwerkfunktionen zuzugreifen. Diese Library funktioniert nur unter Linux und Windows. Sie wird verwendet um auf die Netzwerkinterfaces eines Rechners zuzugreifen und um Netzwerkpakete zu sammeln.

Javax.sound Javax standard Sound Library.

JUnit4: Framework für Softwaretests.

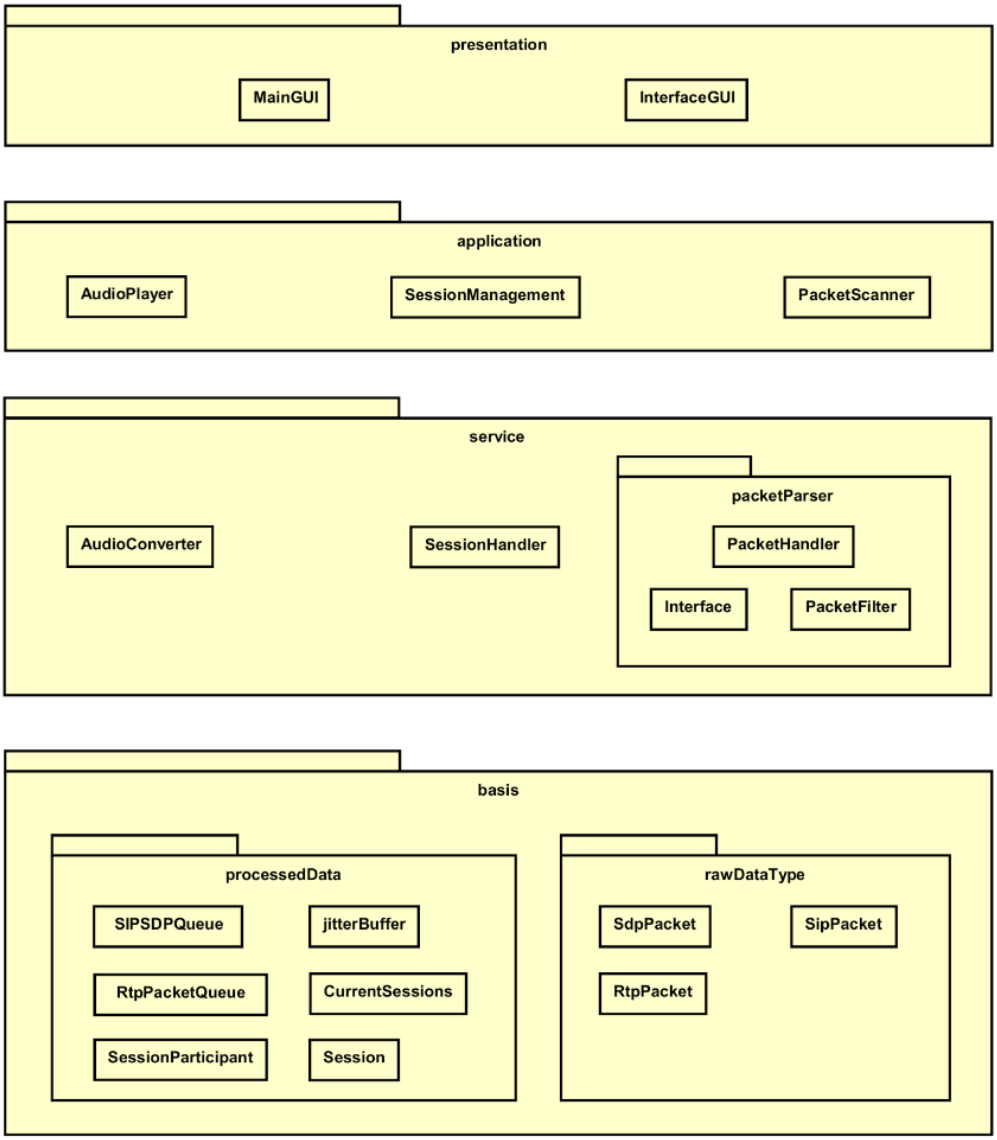
Eclemma: Werkzeug im Eclipse für die Überprüfung der Codeabdeckung von Softwaretests.

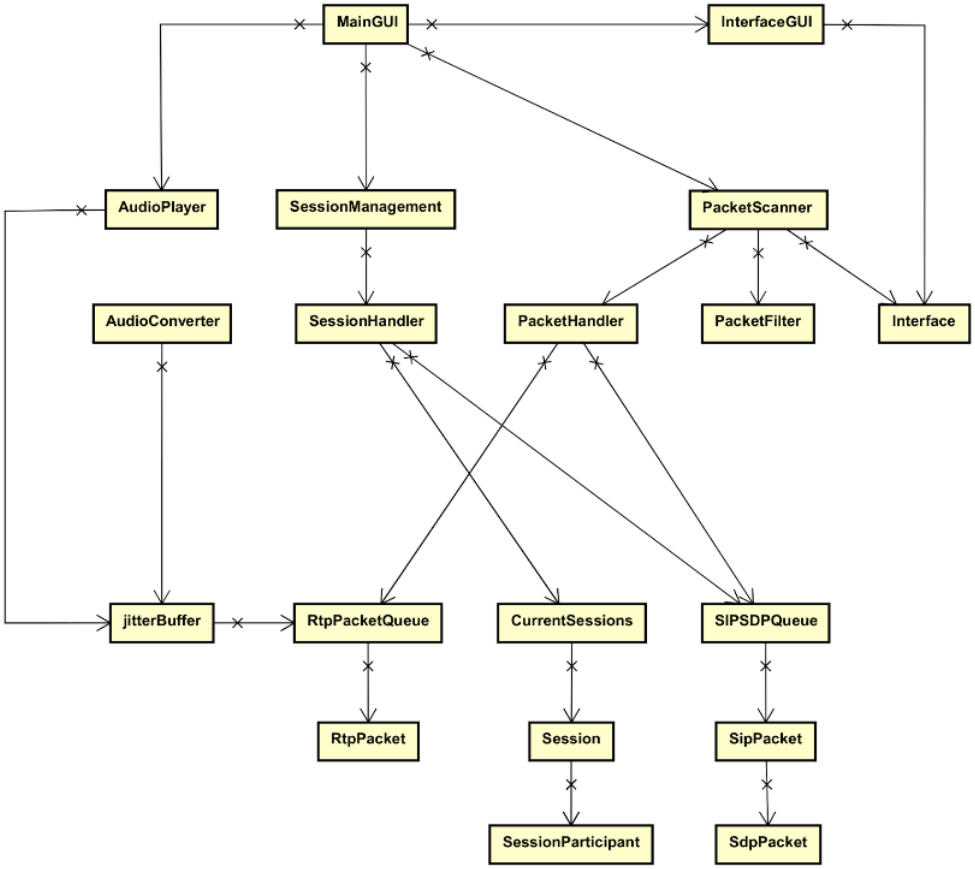
e(fx)clipse: JavaFX plugin für Eclipse

Scene Builder 8.0: Dieses Tool wird verwendet um das GUI zu erstellen.

Travis CI: Tool für continuous integration.

5 Logische Architektur





5.1 Presentation-Layer

Im Präsentation-Layer wird eine MVC (Model, View, Control) Architektur verwendet. So kann das Design von der Logik getrennt werden. Dies bringt den Vorteil, dass zum Beispiel die View ausgetauscht werden kann, ohne dass man etwas an der Logik ändern muss.

5.1.1 Klassenstruktur

Klassenname	Beschreibung
MainGUI	Das MainGUI ist die Hauptoberfläche. Von dieser aus wird alles gestartet und eingestellt. Diese Oberfläche führt auch zum InterfaceGUI.
InterfaceGUI	Im InterfaceGUI werden alle Interfaces angezeigt und man kann das gewünschte Interface bestimmen, auf dem der PacketScanner läuft.

5.1.2 Schnittstellen

Der Presentation-Layer hat eine Schnittstelle zum Application-Layer, sowie zum Service-Layer. Das MainGUI hat eine Schnittstelle zum AudioPlayer, um diesen zu starten und zu stoppen. Zusätzlich besteht noch eine Schnittstelle zum Session-Management, um die Session zu verwalten. Eine weitere Schnittstelle besteht zum PacketScanner, damit dieser gesteuert werden kann, sowie zum InterfaceGUI. Das InterfaceGUI hat eine Schnittstelle zum packetParser Package, welches sich auf dem Service-Layer befindet. Durch diese Schnittstelle kann das aktuelle Interface verwaltet werden.

5.2 Application-Layer

Der Application-Layer beinhaltet die grundlegenden Steuerungsklassen. Durch diesen Layer wird zum Beispiel der AudioPlayer gestartet und gestoppt

5.2.1 Klassenstruktur

Klassenname	Beschreibung
PacketScanner	Der PacketScanner holt alle Pakete vom gewähltem Interface und übergibt diese dem PacketHandler.
AudioPlayer	Der AudioPlayer liest den Buffer aus und spielt diesen Inhalt ab. Dabei gibt wird immer wieder überprüft ob der Buffer bereit ist, damit kein Over- oder Underflow entsteht.
SessionManagement	Mit dem SessionManagement wird die aktuelle Session gesetzt, angepasst und verwaltet.

5.2.2 Schnittstellen

Der Application-Layer greift auf den Service-Layer und auf den Basis-Layer zu. Dieser beinhaltet alle Logikklassen, welche die Pakete verarbeitet und die Sessions erstellt, sowie die zwischengespeicherten Daten. Der PacketScanner greift auf das Package packetParser zu, um die gesnifften Pakete zu verarbeiten. Der AudioPlayer greifft auf den JitterBuffer zu, welche den zu abspielenden Payload beinhaltet. Das SessionManagement greift auf den SessionHandler zu, um die aktuelle Session zu bearbeiten.

5.3 Service-Layer

Die Zuständigkeiten des Service Layers liegen in der Verarbeitung und Erstellung der vom Basis Layer zur Verfügung gestellten Daten.

5.3.1 Packagestruktur

Klassenname	Beschreibung
packetParser	Das packetParser Package beinhaltet den PacketHandler, die Interfaceklasse, sowie den PacketFilter.

5.3.2 Klassenstruktur

Klassenname	Beschreibung
AudioConverter	Der AudioConverter verarbeitet den Audio Payload zu einem abspielbaren Byte Array und füllt dieses in den jitter Buffer.
SessionHandler	Der SessionHandler holt die SIP und SDP Pakete aus der SIPSDPQueue und verarbeitet diese zu neuen Sessions. Falls die Session bereits besteht, wird die richtige Session angepasst.

5.3.3 Schnittstellen

Der Service hat eine Schnittstelle zum Basis-Layer, welche die Datenstruktur beinhaltet. Der SessionHandler hat eine Schnittstelle zum processedData Package, wo die neuen Sessions gespeichert werden, sowie die vorhandenen angepasst werden. Die Schnittstellen des AudioConverters sind auf dem processedData Package. Der AudioConverter holt die Pakete aud der RtpPacketQueue, verarbeitet diese und schreibt den Payload in den JitterBuffer.

5.4 PacketParser-Package

5.4.1 Klassenstruktur

Klassenname	Beschreibung
PacketHandler	Der PacketHandler beinhaltet die Paketlogik. Hier wird jedes Paket überprüft und falls dies ein SIP, SDP oder ein RTP Paket ist, wird es in eine Liste auf dem unteren Layer gespeichert.
Interface	Die Klasse Interface steuert die Interfaceanzeige und Auswahl. Hier werden alle vom Betriebssystem vorhandenen Interface gespeichert und und das aktuell gewünschte Interface gesetzt.
PacketFilter	Im PacketFilter wird der Filter gesetzt, welche Pakete vom PacketScanner bearbeitet werden sollen. In unserem Fall wird hier alles herausgefiltert, was nicht ein UDP-Paket ist.

5.4.2 Schnittstellen

Das parserPacket Package hat eine Schnittstelle zum rawDataType und processedData Package, um die Daten in den Queues zwischenzuspeichern.

5.5 Basis-Layer

Die "basis" Schicht beinhaltet die zu verarbeitenden Daten. Höhere Schichten lesen und schreiben in- und von der basis Schicht aufgrund ihrer eigens definierten Logik.

5.5.1 Klassenstruktur

Package Name	Beschreibung
processedData	Das Package "processed Data" stellt den höher liegenden Schichten die benötigten Daten über fertige Datenstrukturen (Queue, Lists, etc.) zur Verfügung.
rawDataType	Im Package "rawDataType" sind die Implementationen der benötigten Datenpakete. Diese Objekte werden für die Weiterverarbeitung in den höheren Schichten verwendet.

5.5.2 Schnittstellen

Hier bestehen keine Schnittstellen.

5.6 processedData-Package

Das Package "processed Data" stellt den höher liegenden Schichten die benötigten Daten über fertige Datenstrukturen (Queue, Lists, etc.) zur Verfügung.

5.6.1 Klassenstruktur

Klassenname	Beschreibung
Session	Das Sessionobjekt beinhaltet Informationen über die Session (Methoden, Teilnehmer)
SessionParticipant	Das SessionParticipant Objekt beinhaltet die IP-Adresse, SIP-URI und Port
CurrentSessions	Das CurrentSession Objekt speichert die gefundenen Sessions
RtpPacketQueue	Die RtpPacketQueue beinhaltet die zu verarbeitenden RTP Pakete
SIPSDPQueue	Die SIPSDPQueue beinhaltet die zu verarbeitenden SIP/SDP Pakete
JitterBuffer	Der Jitter Buffer beinhaltet den fertigen Audio Payload

5.6.2 Schnittstellen

Hier bestehen keine Schnittstellen.

5.7 RawDataType-Package

Im Package “rawDataType” sind die Implementationen der benötigten Datenpakete. Diese Objekte werden für die Weiterverarbeitung in den höheren Schichten verwendet.

5.7.1 Klassenstruktur

Klassenname	Beschreibung
SipPacket	Dieses Objekt beinhaltet Informationen (z.B. Source, Destination, CallID etc.) über das VoIP Gespräch
SdpPacket	Wird verwendet um Informationen (Ports, Codecs) für die Session zu gewinnen
RtpPacket	Dieses Objekt beinhaltet neben dem Payload noch die Attribute ssrc, seqNr, timeStamp und Port

5.7.2 Schnittstellen

Hier bestehen keine Schnittstellen.

6 Prozesse und Threads

Die Funktionalität der Software macht Parallelität notwendig. Es ist zum Beispiel nicht möglich, dass der Audioplayer mit dem Abspielen des RTP Payloads wartet, bis der Scanner terminiert. Wir setzen auf Multithreading, um den Anforderungen gerecht zu werden.

6.1 Prozesse

Beim Ausführen eines Java Programms wird eine Java Virtual Machine (JVM) gestartet. Die JVM läuft auf einem einzelnen Prozess im Betriebssystem.

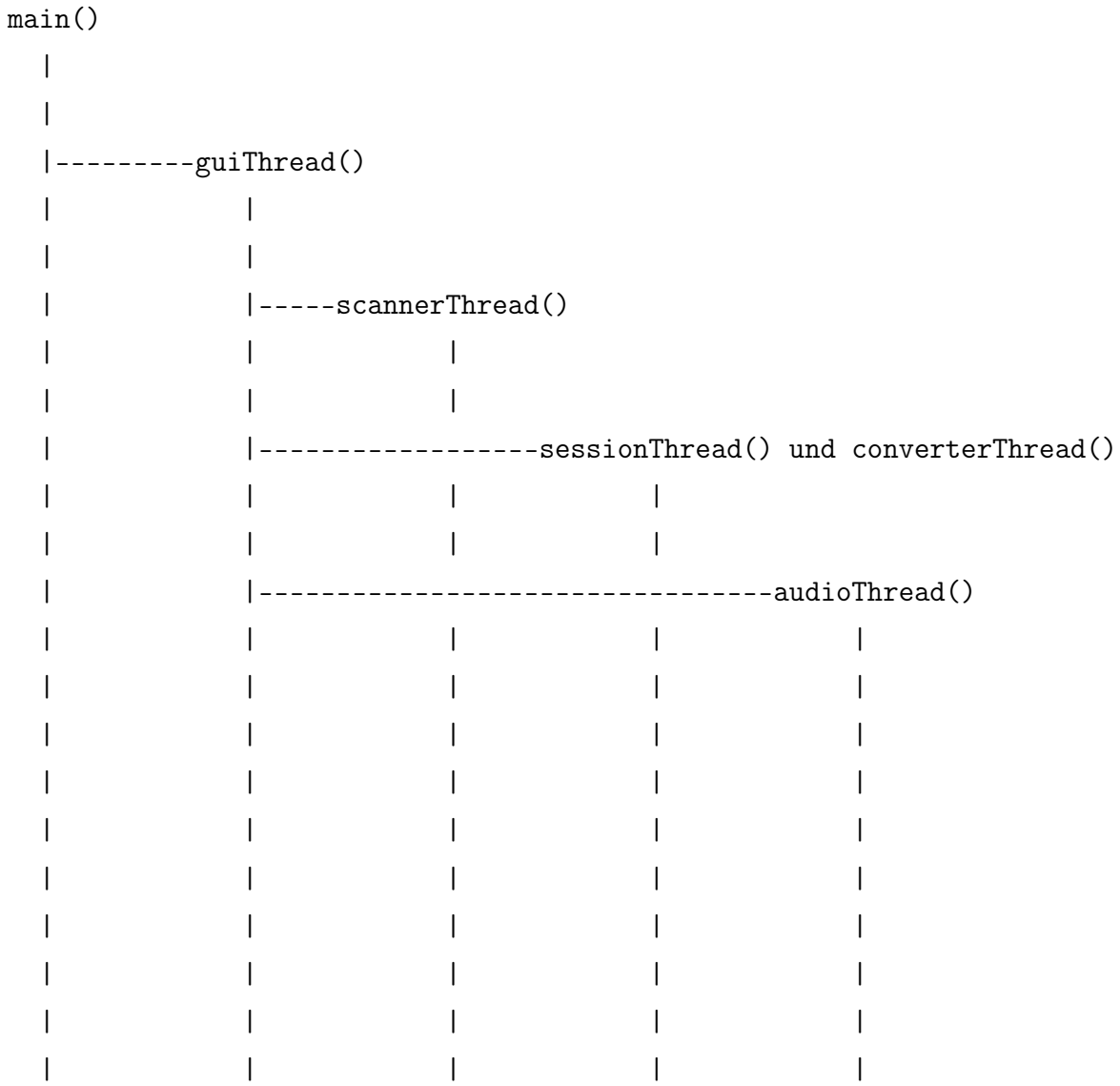
6.2 Threads

Beim Ausführen eines Java Programms ruft ein neu erzeugter Thread die main() - Methode auf. Über die main() - Methode starten wir in einem eigenen Thread das GUI. Weitere für die Programmausführung notwendige Threads werden vom GUI-Thread gestartet. Die Threadumschaltung resp. das Scheduling wird von der JVM selbst übernommen.

6.2.1 Datenstrukturen

Um die Daten zwischenzuspeichern, werden Threadsichere Datenstrukturen verwendet. Dies ist zum Beispiel beim Jitter-Buffer eine ConcurrentLinkedQueue und bei allen anderen Queues eine TransferQueue. Diese sind alle schon in der Standard Java Library vorhanden.

6.2.2 Thread Model



7 Datensicherung

Sniffdatel kommt ohne persistente Datenspeicherung aus. Transistente Daten wie aktuell im Neztwerklaufende VoIP-Session mit den zugehörigen VoIP-Teilnehmern werden in Datenstrukturen gepuffert.

7.1 Jitter-Buffer

Bei der Übertragung von VOIP-Datenpaketen gibt es gewisse Verzögerungen im Netzwerk. Diese Verzögerungen können sehr unterschiedlich sein und werden als Jitter und Delay bezeichnet. Diese Verzögerungen führen zu einer schlechteren Sprachqualität. Um dies zu vermeiden, greift man auf einen Jitter-Buffer zurück. Dieser speichert den eingehenden Verkehr, um gleichmässigen Datenfluss zu garantieren. Mit dem Jitter-Buffer werden vier Probleme behoben. Den Jitter, Packet loss und Out-Of-Order.

7.1.1 Delay

Netzwerkpakete können durch hohe Last auf dem Netzwerk verzögert werden. Dadurch entsteht Delay (Verzögerung). Um diese Verzögerung zu korrigieren, wird der Jitter-Buffer eingesetzt. Dieser puffert die Daten für eine Sekunde in der Queue, damit sicher genügend Pakete vorhanden sind.

7.1.2 Jitter

Jitter ist eigentlich sehr ähnlich wie der Delay. Es handelt sich nur im einen variablen Delay. Somit können wir auch diese Verzögerung mit dem Jitter-Buffer entfernen.

7.1.3 Packet loss

Da Pakete auf dem Weg zum Empfänger verloren gehen können, muss der Packet loss beachtet werden. Fehlen mehr als 1% der Pakete, verschlächtert sich die Soundqualität. Der Packet loss kann sehr leicht durch die Sequenznummer erkannt werden. Um die fehlenden Pakete zu korrigieren, werden durch den AudioConverter 'Null Pakete' hinzugefügt. Diese Pakete enthalten nur Null Bytes und werden vom Player lautlos abgespielt. Dies ist aber ein optinales Feature.

7.1.4 Out-Of-Order

Pakete können aus der Reihenfolge kommen, was die Audioqualität beinträchtigt. Mit Sniffdatel werden als ersten Schritt, die später eintreffenden Pakete verworfen, um die Soundqualität zu steigern. Ein optinales Feature wird sein, die Paketreihenfolge mit dem Jitter-Buffer wiederherzustellen. Dabei wird die Sequenznummer und der Timestamp verwendet.

7.1.5 Datenstruktur

Als Datenstruktur wird eine ConcurrentLinkedQueue verwendet. Diese ist Threadsicher und kann somit von meheren Thread (AudioPlayer, AudioConverter) bearbeitet werden.

8 Grössen und Leistungen

8.1 Leistung

Sniffdatel wird auf eine Datenrate von 100Mbit/s beschränkt. Die Software sollte auf jedem beliebigen Netzwerk, mit der erwähnten Datenrate funktionieren. Der Buffer sowie die Parallelisierung wird dementsprechend integriert.

8.2 Anzahl Sessions

Sniffdatel wird betreffend den anzuzeigenden Sessions auf maximal 50'000 beschränkt. Diese Beschränkung ist nicht kritisch für Sniffdatel, da diese nur temporär in einer Liste gespeichert sind.

8.3 Verzögerung

Als maximale Verzögerung legen wir eine Sekunde fest. Diese Verzögerung betrifft das Zwischenpuffern, damit die Audiodaten flüssig abgespielt werden können.

8.4 Buffergrösse

Sniffdatel kann audio mit dem Codec G.711(µ-law/A-law) wiedergeben. Da dieser Codec mit einer Datenübertragungsrate von 8000Hz*8Bit funktioniert, wird die

Grösse des Jitter-Buffers auf 8KByte (64Kbit/8) festgelegt. Dies entspricht einer Sekunde.