Projektbericht: Entwurf und Implementierung des File-Transfer-Dienstes „myTCP“

Verfasser: Jeremy Braun

Datum: 08.12.25

Lehrveranstaltung: Protocol-Engineering (EK 5060) / Vertiefungspraktikum Rechnernetze (VP 1270)

Betreuer: Prof. Dr. Jürgen Jasperneite, Wolfgang Sonntag

Inhaltsverzeichnis

1. Glossar
2. Aufgabenstellung Protocol-Engineering "myTCP"
3. Anforderungen
    1. Benutzer-Leistungsmerkmale
    2. Technische Leistungsmerkmale
    3. Analyse der Ebenenfunktionen
        1. Anwendungen
        2. Darstellungen
        3. Transportschicht
        4. Vermittlungsschicht
        5. Sicherungsschicht- und Bitübertragungsschicht
    4. Lösungskonzept
        1. Umsetzung der Anforderungen
        2. Abbildung der Anforderungen auf das OSI-Schichtenmodell
            1. Grundlegende Überlegungen zur Schichtenauswahl
            2. Eigenes Schichtenmodell für "myTCP"
        3. Dienst- und Protokollspezifikationen der realisierten Funktionen
            1. A_Layer
            2. Adaption
            3. DL_Layer
    5. Fazit

1. Glossar

Abkürzung	Bedeutung
FTD	**F**ile **T**ransfer **D**ienst
OSI	**O**pen **S**ystem **I**nterconnection
PC	**p**ersonal **C**omputer
MAC	**M**edia **A**ccess **C**ontrol
NIC	**N**etwork **I**nterface **C**ard
GUI	**G**raphical **U**ser **I**nterface
IDE	**I**ntegrated **D**evelopment **E**nvironment
SAP	**S**ervice **A**ccess **P**oint
RX	**R**eceive
TX	**T**ransmit
PDU	**P**rotokoll **D**ata **U**nit
PCI	**P**rotokoll **C**ontroll **I**nformation

2. Aufgabenstellung: Protocol-Engineering "myTCP"

Entwurf und Realisierung eines Dienstes für die zuverlässige Dateiübertragung zwischen zwei PCs.
Randbedingung: Grundsätze des OSI- Schichtenmodell berücksichtigen.

3. Anforderungen

3.1 Benutzer-Leistungsmerkmale

- „.. die Leistungsmerkmale der Datei-Übertragung stichwortartig zusammengestellt, wie sie sich aus der Sicht des PC-Benutzers darstellen sollen. ..“
- Es sollen nur Textdateien übertragen werden können.
- Mit dem Kommunikationssystem erhält der PC-Benutzer einen Dienst, eigene Dateien zu einem der n PCs zu senden, oder aber von einem der n PCs eine Datei zu empfangen.
- Dieser Anwendungsdienst wird im folgenden File-Transfer-Dienst, kurz FTD, genannt.
- Der FTD soll alternativ, aber nur zeitlich nacheinander zum Senden oder zum Empfangen von Textdateien aufrufbar sein (Halbduplex).
- Das Empfangen ist passiv, d.h. ohne Benutzereingaben möglich.
- Das Senden erfordert Eingaben vom Benutzer:
  - Ziel-PC, zu dem eine Datei übertragen werden soll
  - Name der Datei, die übertragen werden soll
  - Ziel-Dateiname, unter dem der übertragene Inhalt auf dem Ziel-PC abgelegt werden soll.
- Die Forderung nach Übertragung von Dateien in einem Netz bedingt eine einheitliche und widerspruchsfreie Adressierung.

- Hierzu sind eindeutige MAC-Adressen und logische Adressen (PC-Namen) der Quell- und Ziel-PCs vorzusehen.
- Der Benutzer soll während der Kommunikation durch Meldungen über den Ablauf der Übertragung informiert werden.
- Der Ziel-PC könnte besetzt oder gar nicht empfangsbereit sein und die Übertragung könnte stark gestört sein, so dass ein Abbruch erforderlich wird.

3.2 Technische Leistungsmerkmale

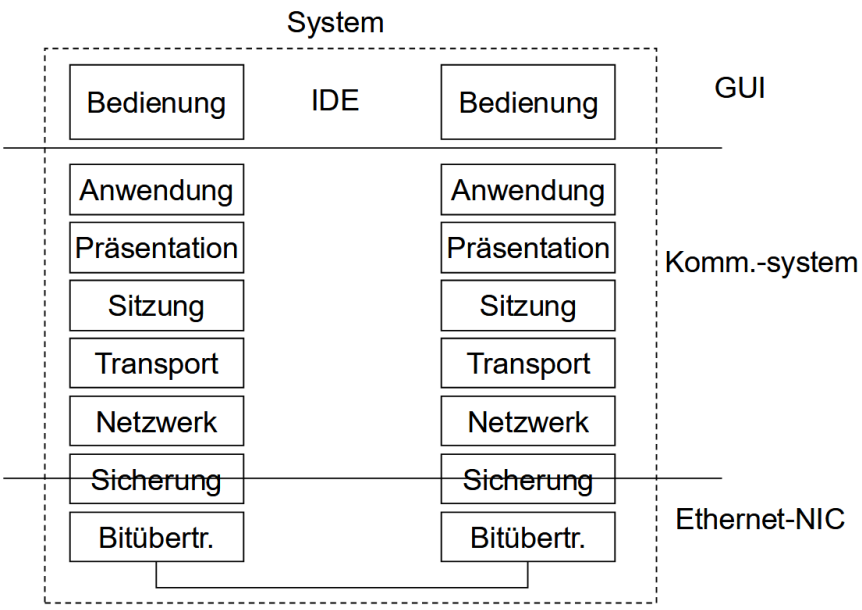
- Auf der Grundlage der bisherigen Festlegungen sollen folgende für die Entwicklung wichtigen Merkmale erfüllt werden:
 - Benutzung der Ethernet-Schnittstelle der PCs
 - Einstellbarkeit notwendiger Kommunikationsparameter
 - Frei wählbare, alphanumerische Namenswahl (logische Adresse) für den PC
 - Herstellen, Halten und Abbauen von temporären Datenverbindungen zwischen je zwei PCs
 - Zuverlässige Übertragung von Telegrammen zwischen den Stationen
 - Übertragung der Dateien in sinnvollen Blöcken (zeilenweise)
 - Korrekte Übertragung von Dateien hinsichtlich Vollständigkeit, Bestätigung an den Benutzer, Wiederaufsetzen bei Übertragungsfehlern.

Rahmenbedingung:

Bei dieser Aufgabenstellung wird auf das Framing (Schicht-2) der Ethernet-Schnittstelle zurückgegriffen. Die Ethernet-Schnittstelle führt auch die Bildung und Auswertung einer Prüfsumme durch.

3.3 Analyse der Ebenenfunktionen

- Welche Funktion gehört in welche Schicht?
- Wie viele Schichten hat unser Kommunikationssystem?



3.3.1 Anwendungsschicht

- Stellt SAP für den File-Transfer-Dienst bereit.
- die Anwendungsschicht benötigt Funktionen, um
  - Dateien öffnen, lesen, schreiben,
  - Senden und Empfangen zeilenweise durchzuführen.
- Verantwortlich für Vollständigkeit der Dateiübertragung
- Zählerstände RX/TX
- Wiederholung bei Ungleichheit (n-mal)

3.3.2 Darstellungsschicht

- Alle PCs
  - INTEL x86, WIN XP SP2 .
  - Dateien bestehen aus ASCII-Zeichen (Codierung/Dekodierung macht Betriebssystem)
  - homogener Rechnerverbund.
- Darstellungsschicht kann entfallen.

3.3.3 Transportschicht

- Unzulänglichkeiten des Kommunikationsnetzes ausgleichen.
- Segmentbildung, Flusskontrolle sind nicht notwendig. Daher ist die Transportschicht im vorliegenden Beispiel leer.
- Transportschicht kann **entfallen**.

3.3.4 Vermittlungsschicht

- Eine Wegewahl (Routing) ist in unserem Beispiel nicht erforderlich.
- Die logische Adressierung (N-Adressen) der Endsysteme soll in alphanumerischer Schreibweise erfolgen.

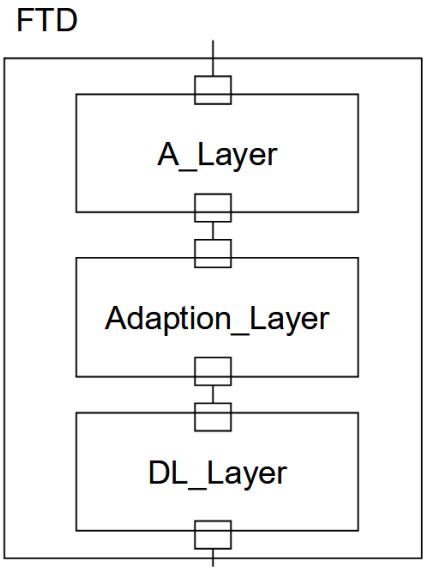
3.3.5 Sicherungs- und Bitübertragungsschicht

- hier IEEE 802.3
- Steuerung des Buszugriffs auf das Übertragungsmedium,
- die Framebildung, Fehler-Sicherung.
- Projektvorgabe: Nutzung der NIC
  - Steuerung des Buszugriffs
  - Fehlersicherung mit Hilfe eines CRC-32
- Die logische Adressierung (N-Adressen) der Endsysteme soll in alphanumerischer Schreibweise erfolgen.
  - Abbildungsfunktion zwischen den logischen Adressen und den physikalischen Adressen (MAC-

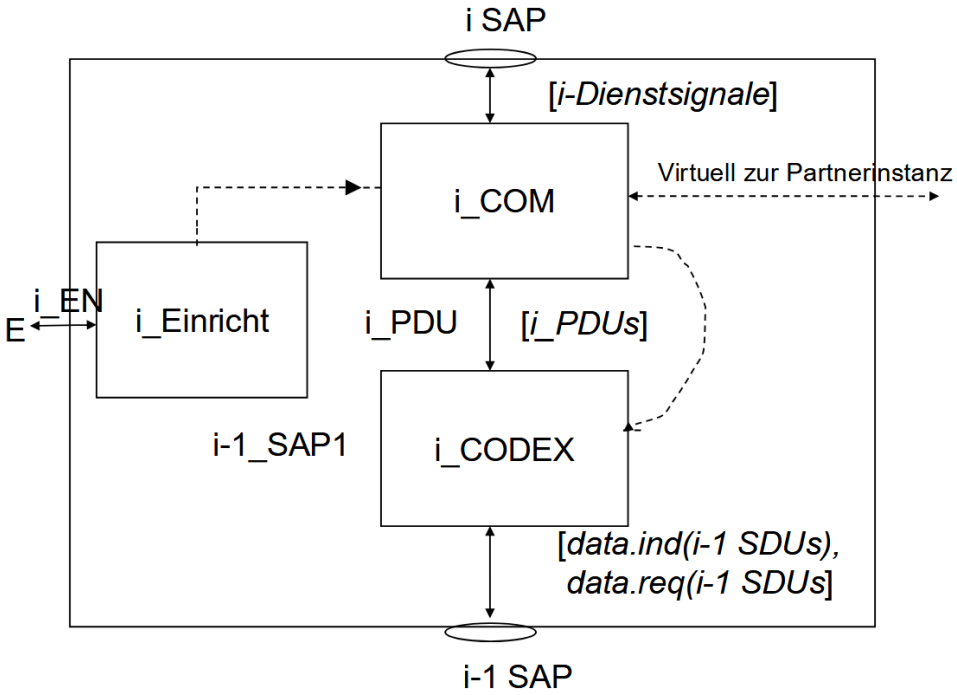
Adressen) ist vorzusehen.

- vgl. ARP

Unser Kommunikationssystem



Genereller Aufbau einer Protokoll-instanz



- i_COM beschreibt das zustandsabhängige Verhalten einer Instanz. (dynamische Protokollspezifikation)
- i_CODEX dient dem Codieren und Dekodieren von PDUs der i-Schicht von und zu Dienstdateneinheiten der i-1 Schicht

4. Lösungskonzept

4.1 Umsetzung der Anforderungen

Als zentrales Werkzeug dieser Umsetzung wurde die Software „IBM Engineering Systems Design Rhapsody“ verwendet. Rhapsody ermöglicht es uns, unser komplexes System durch Nutzung von UML-Standards zu entwerfen, zu analysieren und zu simulieren. Dabei wird aus den grafisch erstellten Diagrammen Code mit hohem Qualitätsstandard generiert.

4.2 Abbildung der Anforderungen auf das OSI-Schichtenmodell

In diesem Kapitel wird erläutert, wie die Anforderungen dieses Projekts auf das siebenschichtige OSI-Referenzmodell abgebildet wurden. Dieses Modell diente als konzeptionelle Grundlage für die zuverlässige Dateiübertragung zwischen zwei PCs. Da jedoch nicht alle Schichten für das gegebene Szenario notwendig sind, wurde eine reduzierte, anwendungsorientierte Schichtenarchitektur entworfen.

4.2.1 Grundlegende Überlegungen zur Schichtenauswahl

Anwendungsschicht

Die Bereitstellung für den Nutzer wurde in Form von aufrufbaren Funktionen realisiert.

Darstellungsschicht

Es wurde keine Form der Datenkomprimierung oder -formatierung implementiert. Die Kodierung der Nachrichten erfolgt in stark vereinfachter Form im ACodex (siehe 4.3.1).

Sitzungsschicht

Im A-Layer wurde eine Halbduplex-Steuerung in vereinfachter Form integriert.

Transportschicht

Bei der Übertragung wird die .txt-Datei in zeilenweise aufgeteilte Strings übertragen. Anschließend wird geprüft, ob die Anzahl der übertragenen Zeilen mit der Anzahl der empfangenen Zeilen übereinstimmt.

Vermittlungsschicht

Die logische Adressierung wird durch Eingabe der PC-Namen realisiert, aber es findet kein Routing in andere Netzwerke statt. Somit kann die Kommunikation mit einem anderen PC nur erfolgen, wenn sich beide im selben Netzwerk befinden.

Sicherungsschicht

Nach einem initialen Broadcast wird die MAC-Adresse des Übertragungspartners gespeichert und in den darauffolgenden Frames genutzt. Die Ethernet-Hardware übernimmt die Fehlererkennung (CRC), jedoch keine Bestätigungen oder Wiederholungen – dies obliegt den höheren Schichten.

Bitübertragungsschicht

Durch Funktionen, die von Herrn Wolfgang Sonntag bereitgestellt wurden, können wir die Ethernet-Hardware der Netzwerkkarte nutzen, Frames zum Senden übergeben und prüfen, ob neue Frames angekommen sind.

4.2.2 Eigenes Schichtenmodell für "myTCP"

Auf Basis dieser Analyse wurde ein dreischichtiger Protokoll-Stack entworfen, der die OSI-Funktionalitäten bündelt:

A_Layer
Das A_Layer fasst die Schichten 7–4 (Anwendung, Darstellung, Sitzung, Transport) grob zusammen und:
- stellt einen SAP für den FTD bereit,
- öffnet, liest und schreibt Dateien,
- steuert die Halbduplex-Verbindung,
- kontrolliert die Vollständigkeit der Übertragung mittels RX-/TX-Zählern,
- löst Timeouts bei ausbleibenden Antworten aus.

Adaption_Layer
Das Adaption_Layer hat keine eigene Funktionalität und reicht die ankommenden Signale des A_Layer an das DL-Layer weiter und umgekehrt. Dieses Layer dient als Platzhalter, um das Modell erweitern zu können, ohne die SAPs des A_Layer oder DL_Layer anpassen zu müssen.

DL-Layer
Das DL_Layer fasst die übrigen Schichten 3–1 zusammen (Vermittlung, Sicherung, Bitübertragung) und:
- verwaltet die Zuordnung der PC-Namen zu MAC-Adressen,
- bildet PC-Namen auf MAC-Adressen ab,
- gibt Frames an die NIC des PCs weiter,
- prüft regelmäßig, ob neue Frames angekommen sind.

4.3 Dienst- und Protokollspezifikationen der realisierten Funktionen

4.3.1 A_Layer

Um die Funktionen des A_Layer übersichtlich zu strukturieren, wurde er logisch in ACodex und ACom getrennt. Der ACodex übernimmt die En- und Dekodierung der Nachrichten, der ACom stellt den Gesamtablauf und die Behandlung von Sonderfällen in einem Zustandsautomaten dar.

ACom

Der Zustandsautomat beginnt im Zustand „Idle“. Von hier aus kann er entweder in den Send-Modus wechseln oder über eine Connect-Indication in den Empfangspfad übergehen.

Sendepfad
Im Send-Modus wartet der Automat auf einen `sendFile`-Befehl mit den notwendigen Informationen (Ziel-PC, Quell-Dateiname, Ziel-Dateiname). Nach dem Erhalt wird ein Connect-Request (`CNReq`) zur Kodierung an den ACodex weitergegeben. Erfolgt innerhalb von drei Sekunden keine Bestätigung (`CNCnf`), kehrt der Automat in den Idle zurück. Nach erfolgreichem Verbindungsaufbau sendet der Automat zuerst den Dateinamen (`Filename`) und anschließend die Textdatei Zeile für Zeile (`Transmit`). Sobald die letzte Zeile übertragen wurde, wird ein Disconnect-Request (`DCNReq`) mit der Anzahl der gesendeten Zeilen an den ACodex übergeben. Nun erwartet der Automat eine Bestätigung des Verbindungsabbaus (`DCNCnf`) zusammen mit der Information, ob alle Zeilen korrekt angekommen sind („+“) oder nicht („-“). Bei einem Timeout von drei Sekunden oder bei einer negativen Bestätigung kehrt der Automat ebenfalls in den Idle zurück.

Empfangspfad
Erhält der Automat im Idle eine Connect-Indication (`CNInd`), so erwidert er den Verbindungsversuch mit einem Connect-Response (`CNRes`) und wartet auf den Dateinamen sowie die folgenden Zeilen der Textdatei. Dabei zählt er die empfangenen Zeilen und schreibt die Datei. Sobald eine Disconnect-Indication (`DCNInd`) empfangen wird, prüft der Automat, ob die Anzahl der empfangenen Zeilen mit der in der DCNInd mitgelieferten Zahl übereinstimmt, und sendet einen Disconnect-Response (`DCNRes`) mit „+“ bei Identität bzw. „-“ bei Abweichung.

ACodex

Der ACodex kodiert jede Anfrage und Antwort des ACom in eine PDU. Diese wird in einen `PDataReq` verpackt und an den Adaption-Layer weitergegeben. Wird eine `PDataInd` empfangen, so wird der Inhalt dekodiert und der entsprechende Befehl im ACom ausgelöst.

4.3.2 Adaption

Der Adaption-Layer besteht aus einem Zustand „Idle“ und kann auf zwei Aktionen reagieren:

1. Ruft der A_Layer die Funktion `PDataReq` auf, so wird der Payload in einen `DLDataReq` verpackt und an das DL_Layer (genauer: an den DLCom) weitergegeben.
2. Ruft der DL_Layer die Funktion `DLDataInd` auf, so wird der Payload in eine `PDataInd` verpackt und an den ACodex weitergeleitet.

4.3.3 DL_Layer

DLCom

Der Zustandsautomat beginnt im Idle und kann logisch in Sende- und Empfangspfad unterteilt werden.

Sendepfad
Erhält der Automat einen `DLDataReq` vom Adaption-Layer, wird geprüft, ob es sich um einen Verbindungsaufbau (`CNReq`) handelt. In diesem Fall wird ein `Transmit`-Signal an den DLCodex mit der Broadcast-Adresse (`ff-ff-ff-ff-ff-ff`) ausgelöst. Handelt es sich um eine andere Nachricht (z. B. `Filename`, `Transmit`, `DCNReq`), wird die zwischengespeicherte MAC-Adresse des Kommunikationspartners verwendet und ein Unicast ausgelöst.

Empfangspfad
Bei einem Eingangssignal (`RX`) vom DLCodex wird zuerst geprüft, für wen das Paket bestimmt ist. Ist die Ziel-MAC-Adresse nicht die eigene und nicht der Broadcast, wird das Paket verworfen. Bei einem Broadcast wird die PDU dekodiert und geprüft, ob der eigene PC-Name als Ziel genannt ist. Ist dies der Fall (z. B. bei einem Connect-Request), wird die MAC-Adresse des Senders zwischengespeichert und eine `DLDataInd` mit dem Payload an den Adaption-Layer gesendet. Bei einem Unicast an die eigene Adresse wird der Payload ebenfalls per `DLDataInd` weitergereicht.

DLCodex

- Der DLCodex realisiert die Kommunikation mit dem Physical Layer (NIC).
  Erhält er vom DLCom ein `Transmit`-Signal, so verpackt er die Nachricht in einen Ethernet-Frame (mit Ziel-MAC, Quell-MAC und Payload) und übergibt diesen zum Versenden an die NIC.
- In regelmäßigen Intervallen prüft er, ob an der NIC ein neuer Frame angekommen ist. Ist dies der Fall, wird der Frame ausgelesen, der Payload extrahiert und ein `RX`-Signal beim DLCom ausgelöst.

5. Fazit

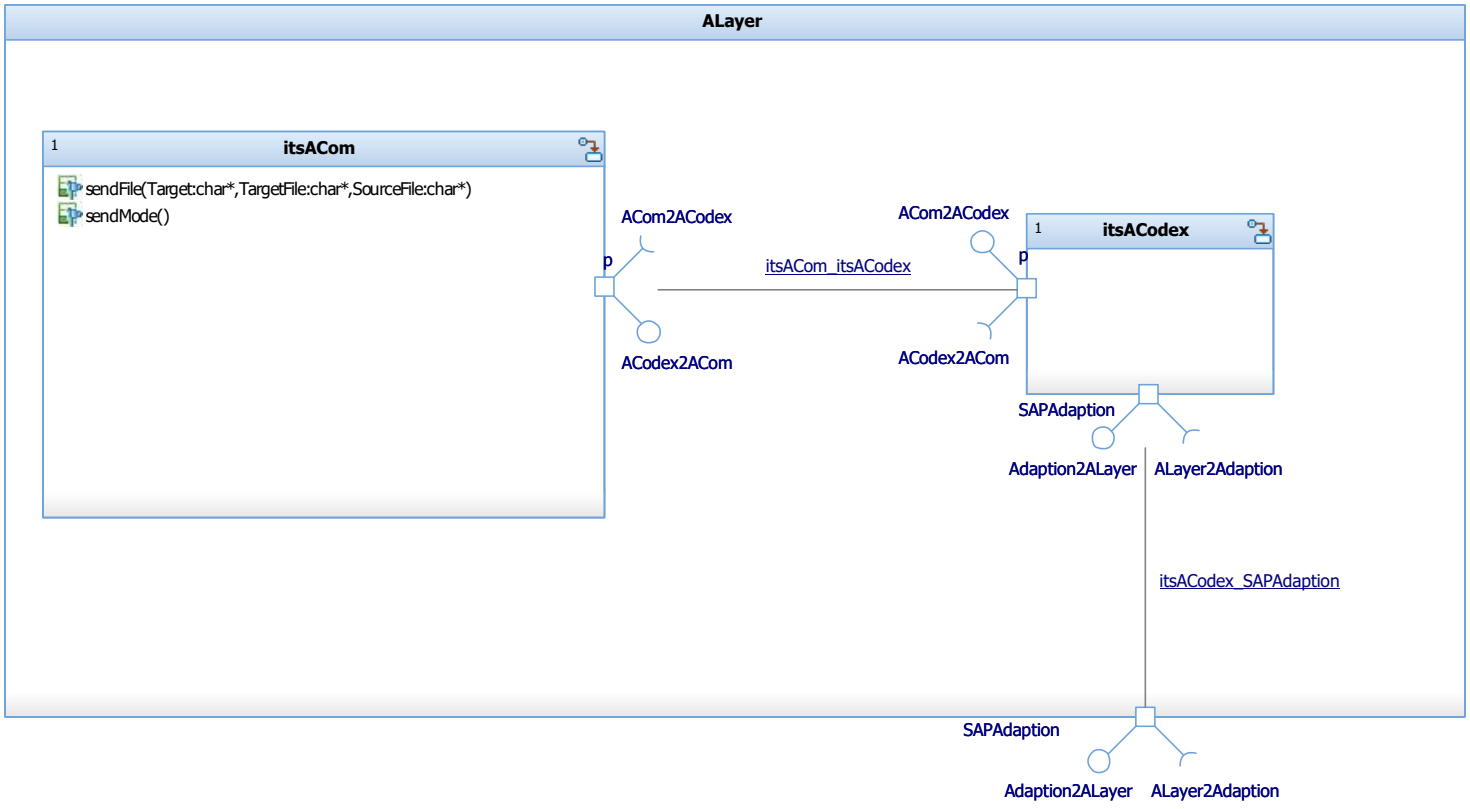
Im Rahmen dieser Projektarbeit wurde der File-Transfer-Dienst „myTCP“ erfolgreich entworfen und implementiert. Ausgehend von den definierten Benutzer- und technischen Leistungsmerkmalen erfolgte eine systematische Analyse der Anforderungen und deren Abbildung auf ein an das OSI-Schichtenmodell angelehntes, dreischichtiges Kommunikationssystem. Dieses besteht aus dem Anwendungs-Layer (A_Layer), einem rein vermittelnden Adaption_Layer und dem Data-Link-Layer (DL_Layer), das die Funktionen der Bitübertragung, Sicherung und Vermittlung bündelt.
Die Dienst- und Protokollspezifikationen wurden mit Zustandsautomaten (ACom, DLCom) und Kodierern (ACodex, DLCodex) in IBM Rhapsody modelliert. Die erstellten Sequenzdiagramme belegen die grundlegende Funktionsfähigkeit der implementierten Abläufe für Verbindungsaufbau, dateiweise Übertragung und Verbindungsabbau. Der Dienst ermöglicht es dem Benutzer, in einem halbduplexen Verfahren Textdateien zwischen zwei PCs zu senden oder zu empfangen, wobei er über Meldungen zum aktuellen Stand informiert wird.
Ein kritischer Punkt ist die Umsetzung der technischen Leistungsmerkmale. Die Forderung nach einem **„Wiederaufsetzen bei Übertragungsfehlern“ wurde in der aktuellen Implementierung nicht realisiert**. Zwar wird die Vollständigkeit der Übertragung durch einen Zeilenabgleich am Ende überprüft und ein negativer Abschluss signalisiert, eine automatische Wiederholung fehlerhafter oder verlorengegangener Blöcke findet jedoch nicht statt. Hier besteht Potenzial für zukünftige Erweiterungen, um die Zuverlässigkeit des Dienstes weiter zu erhöhen.
Insgesamt konnte mit „myTCP“ ein funktionsfähiger Prototyp für einen simplen File-Transfer-Dienst entwickelt werden, der die grundlegenden Konzepte des Protocol-Engineerings praktisch anwendet und demonstriert.

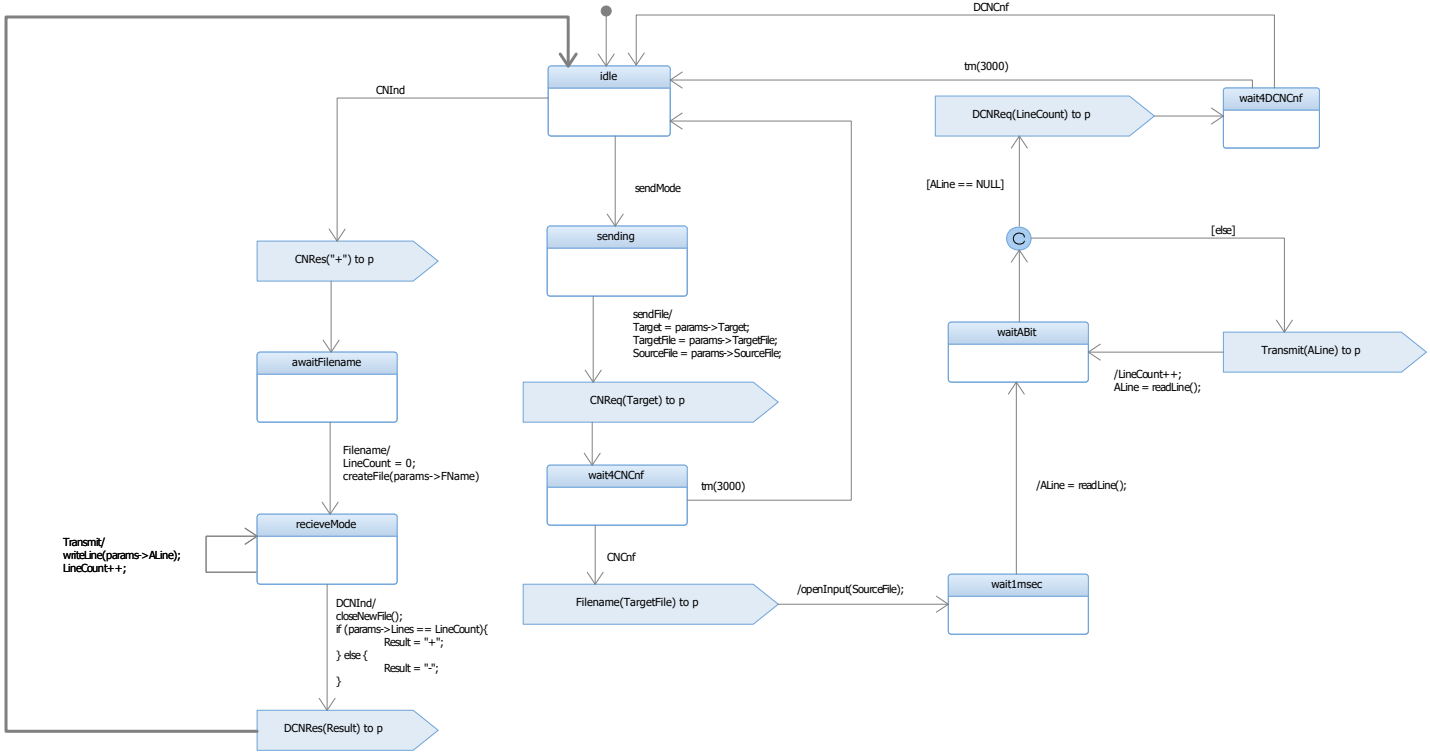
Eidesstattliche Erklärung

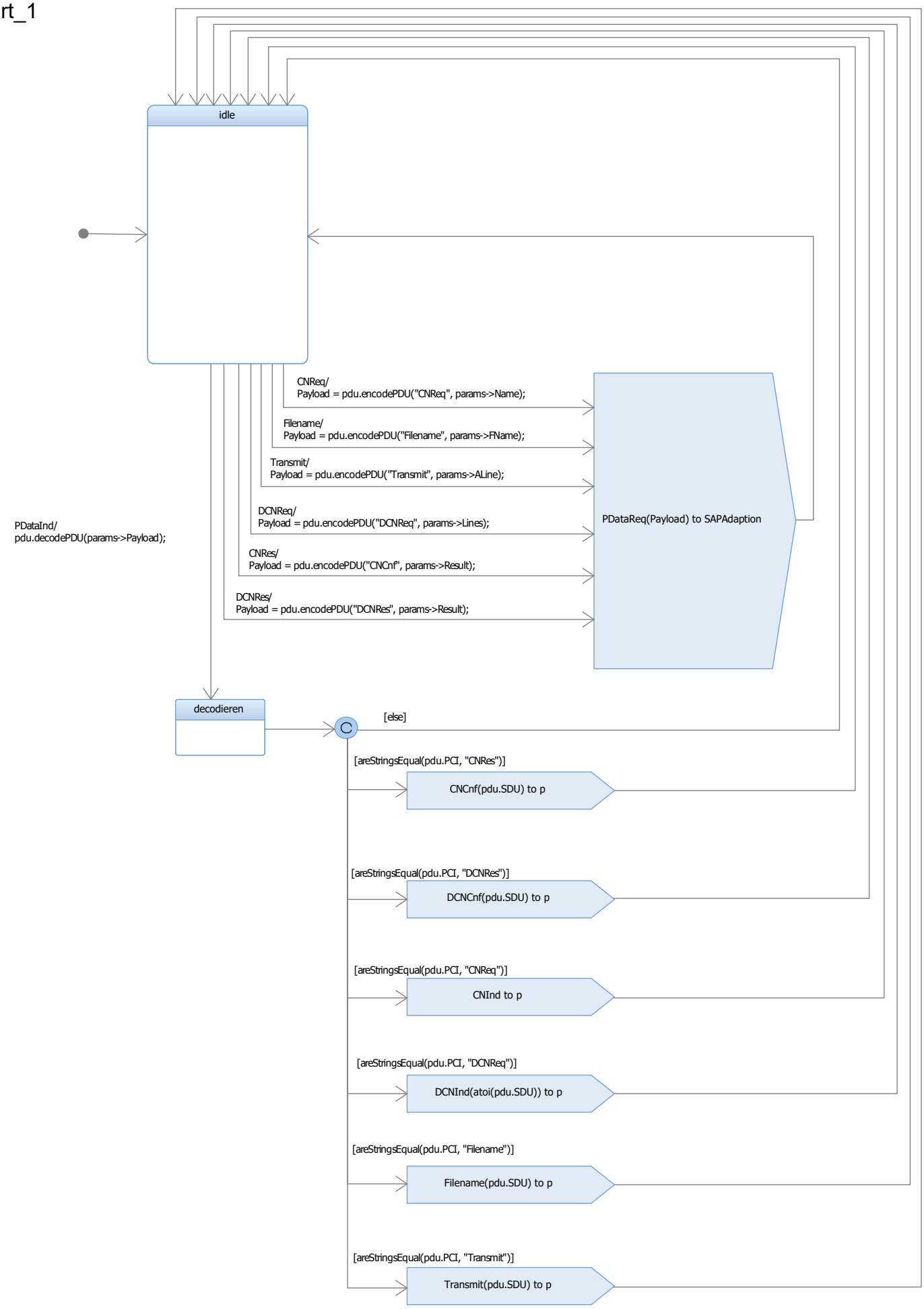
Ich erkläre hiermit an Eides statt, dass ich die vorliegende Ausarbeitung selbstständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

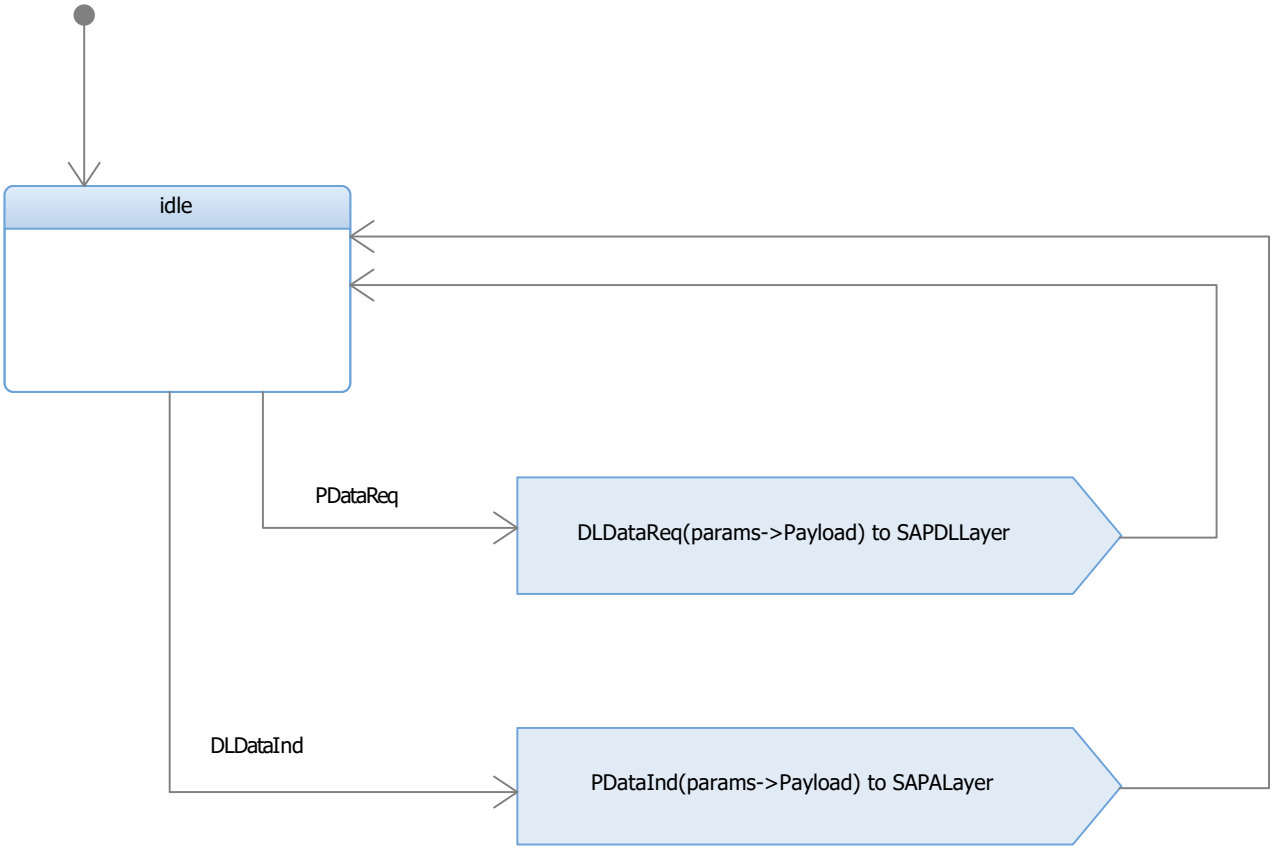
Detmold, den 27.02.26

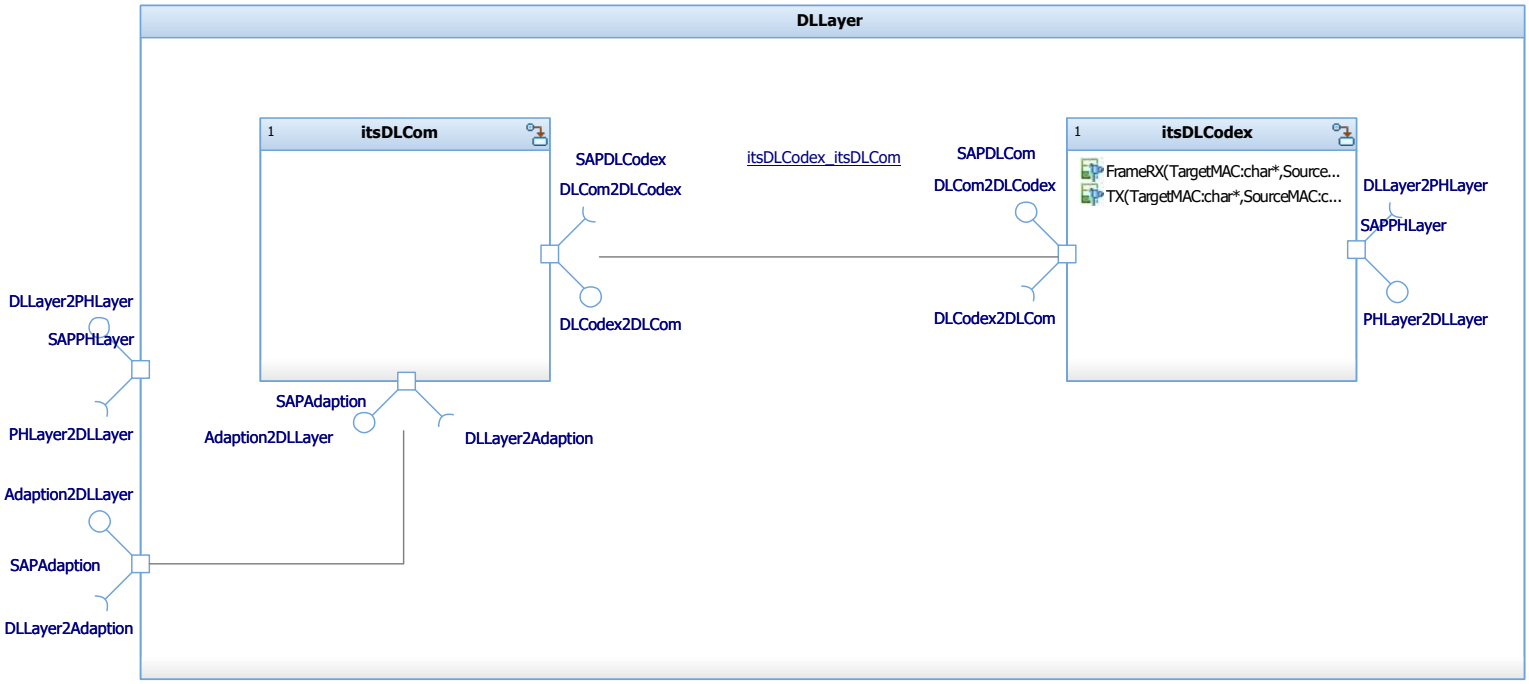
(Unterschrift)


ALayer
1
itsACom
sendFile(Target:char*,TargetFile:char*,SourceFile:char*)
sendMode()
ACom2ACodex
p
ACodex2ACom
itsACom_itsACodex
ACom2ACodex
p
ACodex2ACom
1
itsACodex
SAPAdaption
Adaption2ALayer
ALayer2Adaption
itsACodex_SAPAdaption
SAPAdaption
Adaption2ALayer
ALayer2Adaption

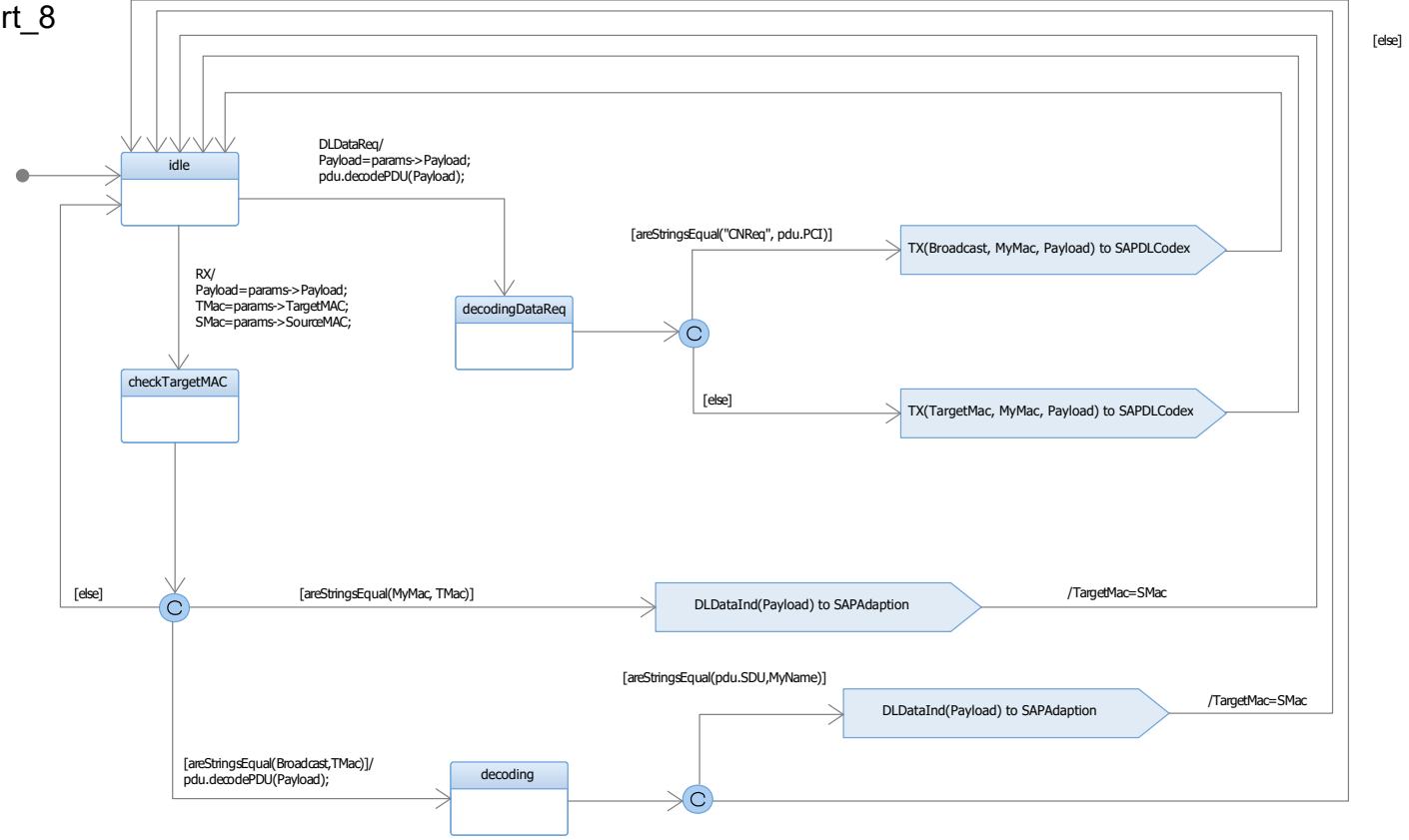

DCNCnf
idle
tm(3000)
wait4DCNCnf
CNInd
DCNReq(LineCount) to p
sendMode
[ALine == NULL]
sending
[else]
CNRes("+") to p
sendFile/
Target = params->Target;
TargetFile = params->TargetFile;
SourceFile = params->SourceFile;
waitABit
Transmit(ALine) to p
awaitFilename
/LineCount++;
ALine = readLine();
CNReq(Target) to p
Filename/
LineCount = 0;
createFile(params->FName)
wait4CNCnf
tm(3000)
/ALine = readLine();
recieveMode
Transmit/
writeLine(params->ALine);
LineCount++;
CNCnf
Filename(TargetFile) to p
/openInput(SourceFile);
wait1msec
DCNInd/
closeNewFile();
if (params->Lines == LineCount){
Result = "+";
} else {
Result = "-";
}
DCNRes(Result) to p


idle
CNReq/
Payload = pdu.encodePDU("CNReq", params->Name);
Filename/
Payload = pdu.encodePDU("Filename", params->FName);
Transmit/
Payload = pdu.encodePDU("Transmit", params->ALine);
DCNReq/
Payload = pdu.encodePDU("DCNReq", params->Lines);
CNRes/
Payload = pdu.encodePDU("CNCnf", params->Result);
DCNRes/
Payload = pdu.encodePDU("DCNRes", params->Result);
PDataReq(Payload) to SAPAdaption
PDataInd/
pdu.decodePDU(params->Payload);
decodieren
[else]
[areStringsEqual(pdu.PCI, "CNRes")]
CNCnf(pdu.SDU) to p
[areStringsEqual(pdu.PCI, "DCNRes")]
DCNCnf(pdu.SDU) to p
[areStringsEqual(pdu.PCI, "CNReq")]
CNInd to p
[areStringsEqual(pdu.PCI, "DCNReq")]
DCNInd(atoi(pdu.SDU)) to p
[areStringsEqual(pdu.PCI, "Filename")]
Filename(pdu.SDU) to p
[areStringsEqual(pdu.PCI, "Transmit")]
Transmit(pdu.SDU) to p

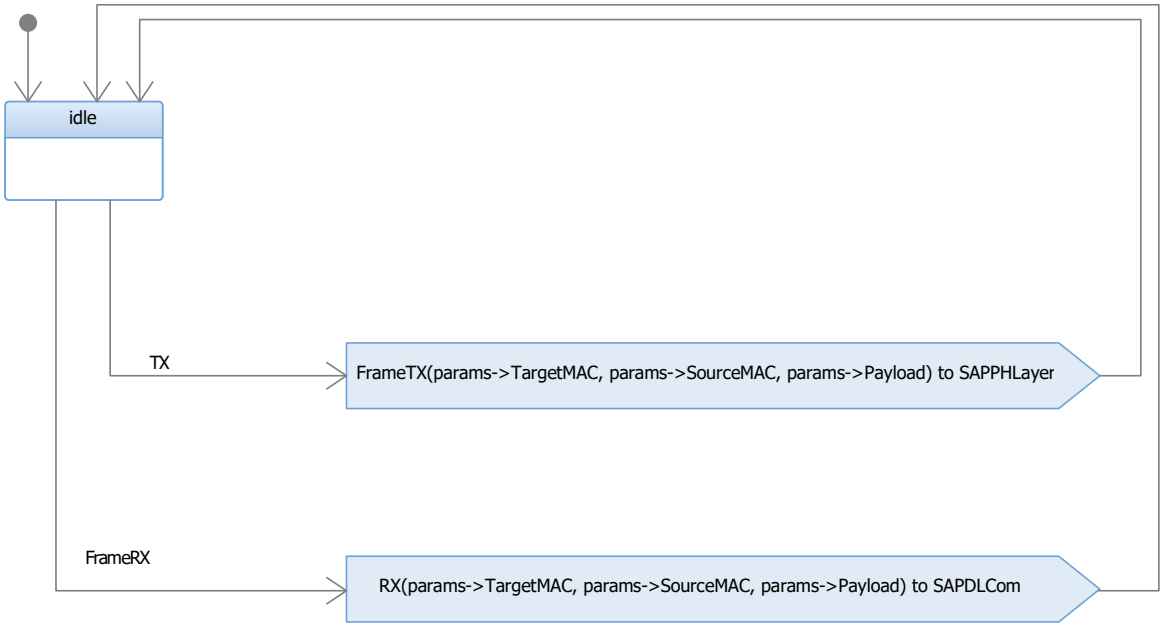

idle
PDataReq
DLDataReq(params->Payload) to SAPDLLayer
DLDataInd
PDataInd(params->Payload) to SAPALayer

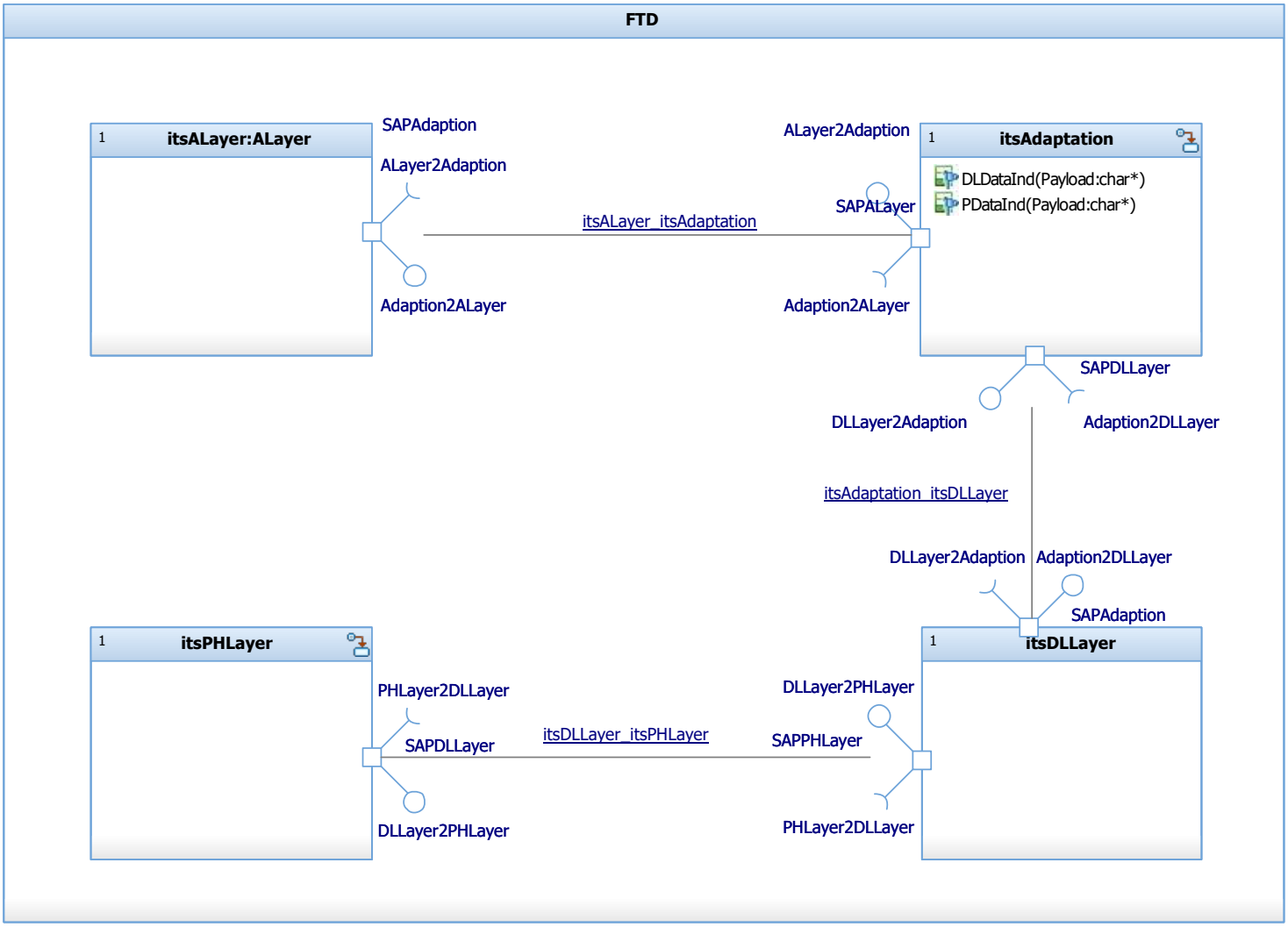

DLLayer
1
itsDLCom
SAPDLCodex
DLCom2DLCodex
DLCodex2DLCom
itsDLCodex_itsDLCom
SAPDLCom
DLCom2DLCodex
DLCodex2DLCom
1
itsDLCodex
FrameRX(TargetMAC:char*,Source...
TX(TargetMAC:char*,SourceMAC:c...
DLLayer2PHLayer
SAPPHLayer
PHLayer2DLLayer
DLLayer2PHLayer
SAPPHLayer
PHLayer2DLLayer
SAPAdaption
Adaption2DLLayer
DLLayer2Adaption
Adaption2DLLayer
SAPAdaption
DLLayer2Adaption

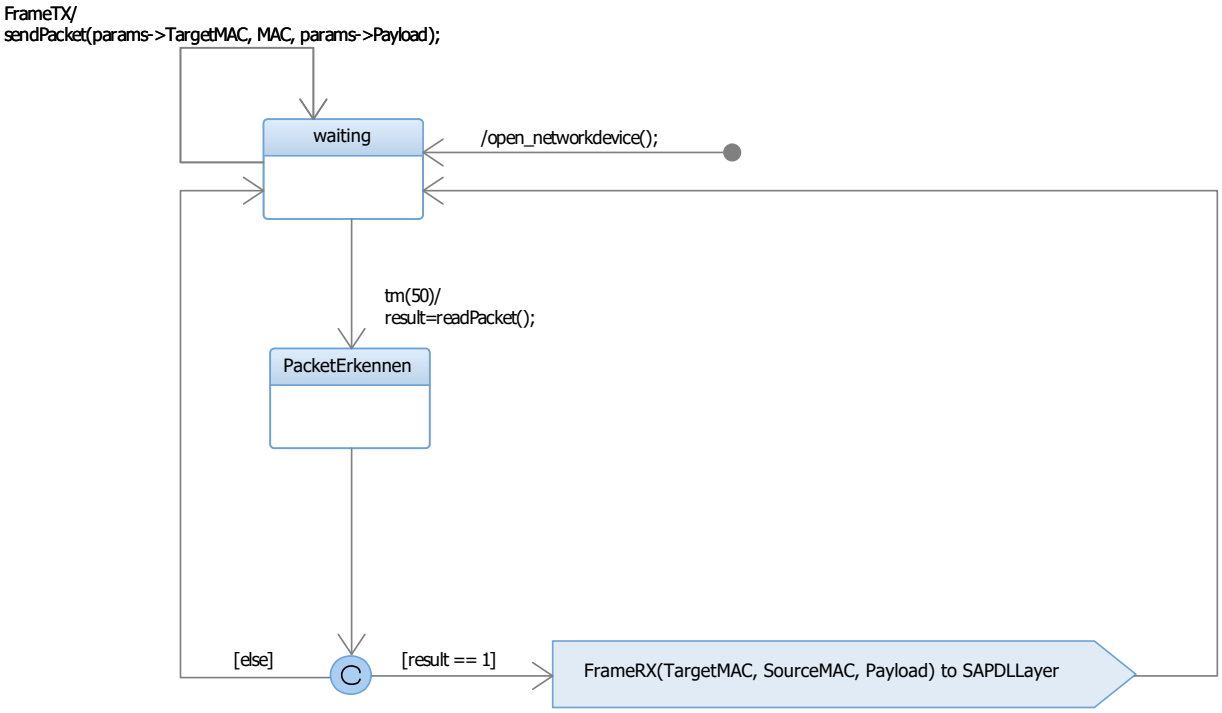
statechart_8

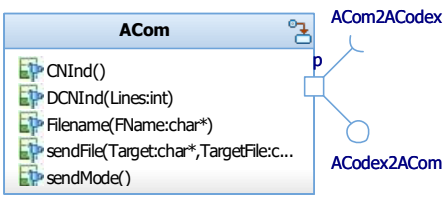


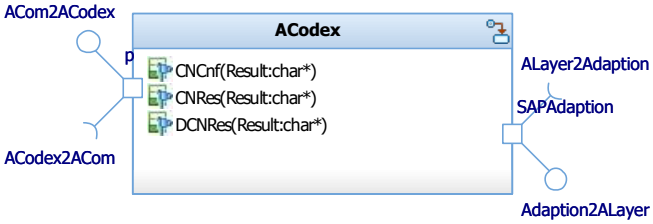
statechart_1

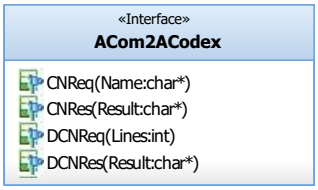


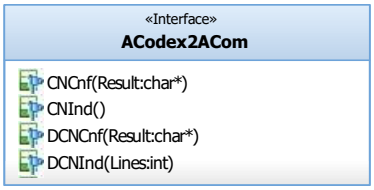

FTD
1
itsALayer:ALayer
SAPAdaption
ALayer2Adaption
Adaption2ALayer
itsALayer_itsAdaptation
ALayer2Adaption
SAPALayer
Adaption2ALayer
1
itsAdaptation
DLDataInd(Payload:char*)
PDataInd(Payload:char*)
SAPDLLayer
DLLayer2Adaption
Adaption2DLLayer
itsAdaptation_itsDLLayer
DLLayer2Adaption
Adaption2DLLayer
SAPAdaption
1
itsPHLayer
PHLayer2DLLayer
SAPDLLayer
DLLayer2PHLayer
itsDLLayer_itsPHLayer
DLLayer2PHLayer
SAPPHLayer
PHLayer2DLLayer
1
itsDLLayer


FrameTX/
sendPacket(params->TargetMAC, MAC, params->Payload);
waiting
/open_networkdevice();
tm(50)/
result=readPacket();
PacketErkennen
[else]
C
[result == 1]
FrameRX(TargetMAC, SourceMAC, Payload) to SAPDLLayer

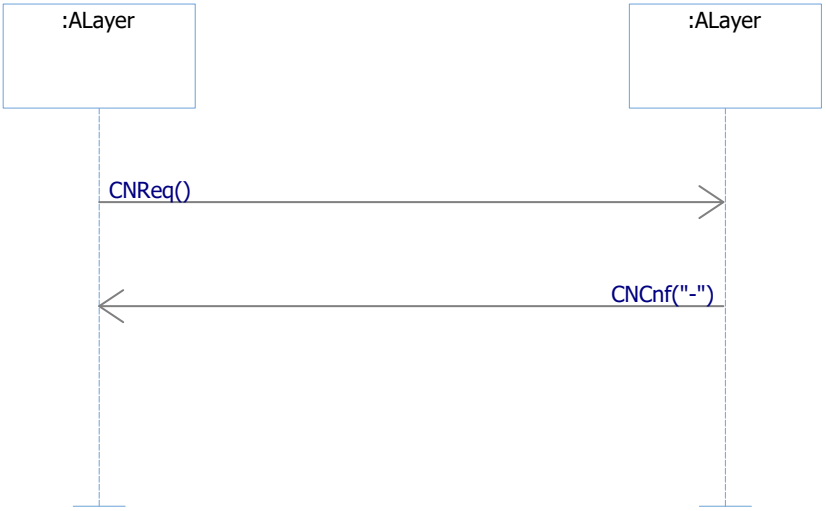

ACom
CNInd()
DCNInd(Lines:int)
Filename(FName:char*)
sendFile(Target:char*,TargetFile:c...
sendMode()
p
ACom2ACodex
ACodex2ACom


ACom2ACodex
p
ACodex2ACom
ACodex
CNCnf(Result:char*)
CNRes(Result:char*)
DCNRes(Result:char*)
ALayer2Adaption
SAPAdaption
Adaption2ALayer

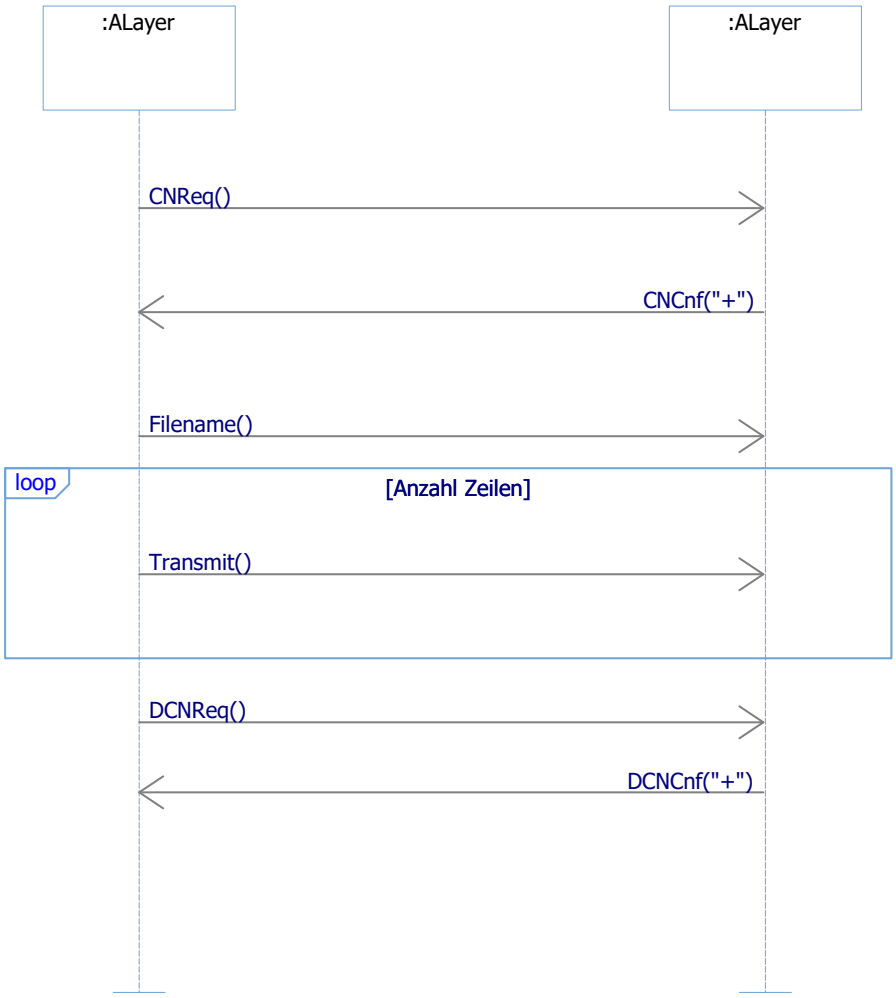

«Interface»
ACom2ACodex
CNReq(Name:char*)
CNRes(Result:char*)
DCNReq(Lines:int)
DCNRes(Result:char*)


«Interface»
ACodex2ACom
CNCnf(Result:char*)
CNInd()
DCNCnf(Result:char*)
DCNInd(Lines:int)

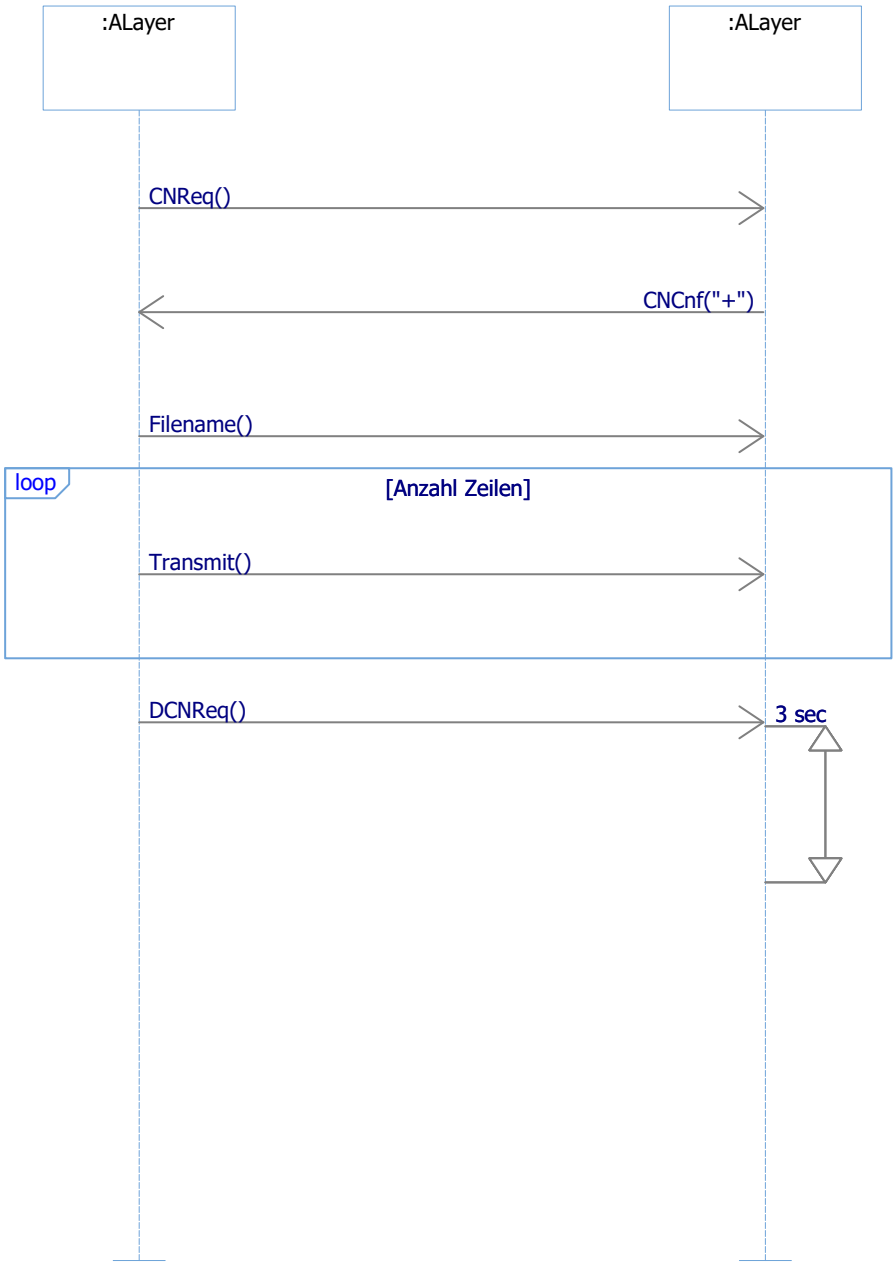
CNCnf schlaegt 1mal fehl



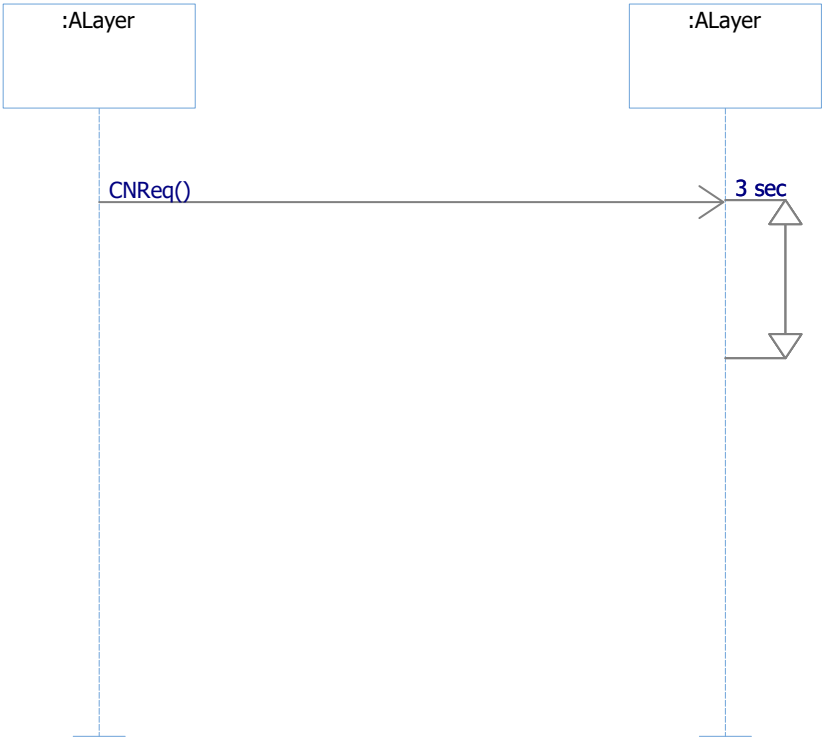
Erfolgreicher Ablauf



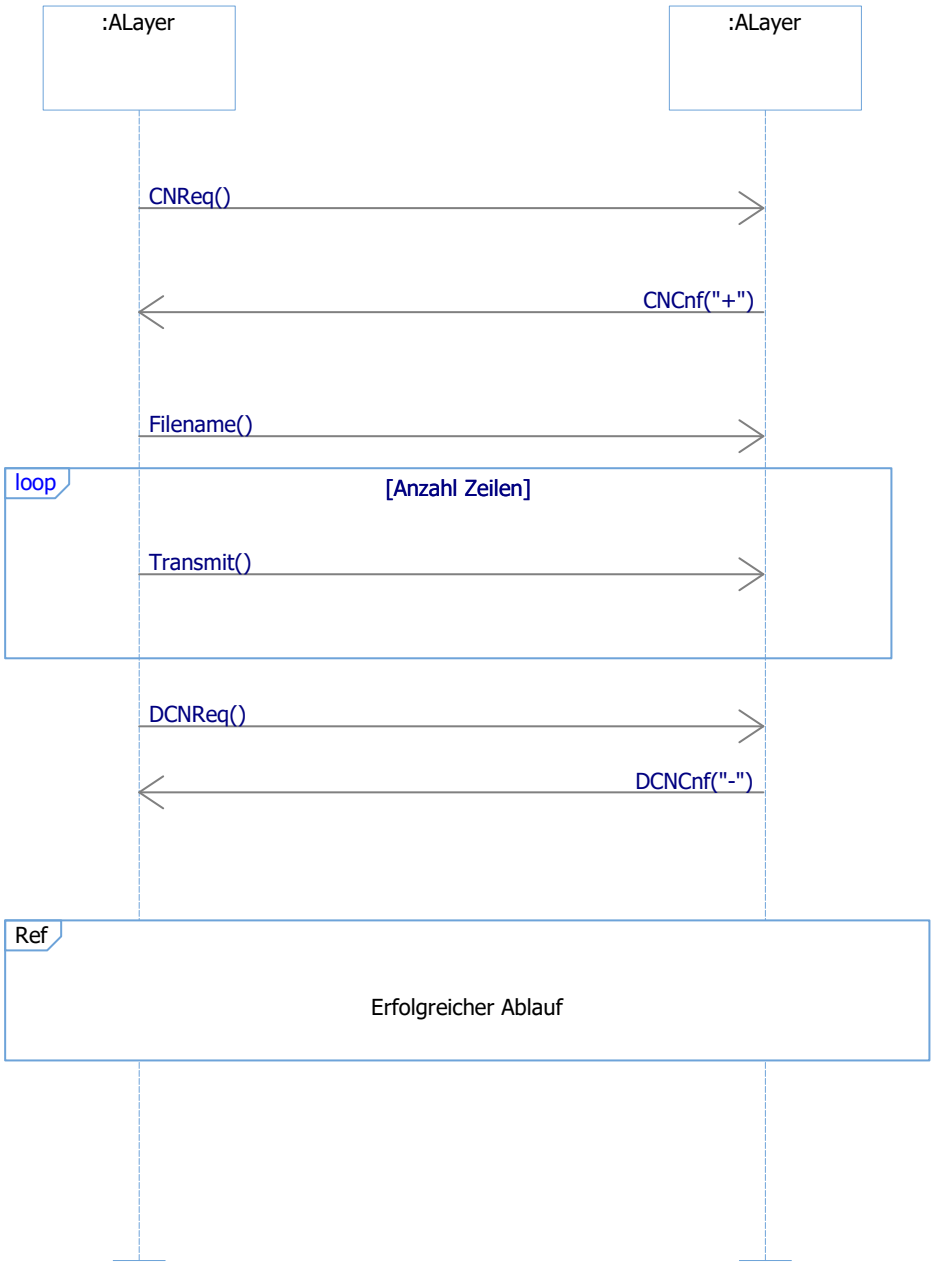
DCNCnf schlaegt fehl



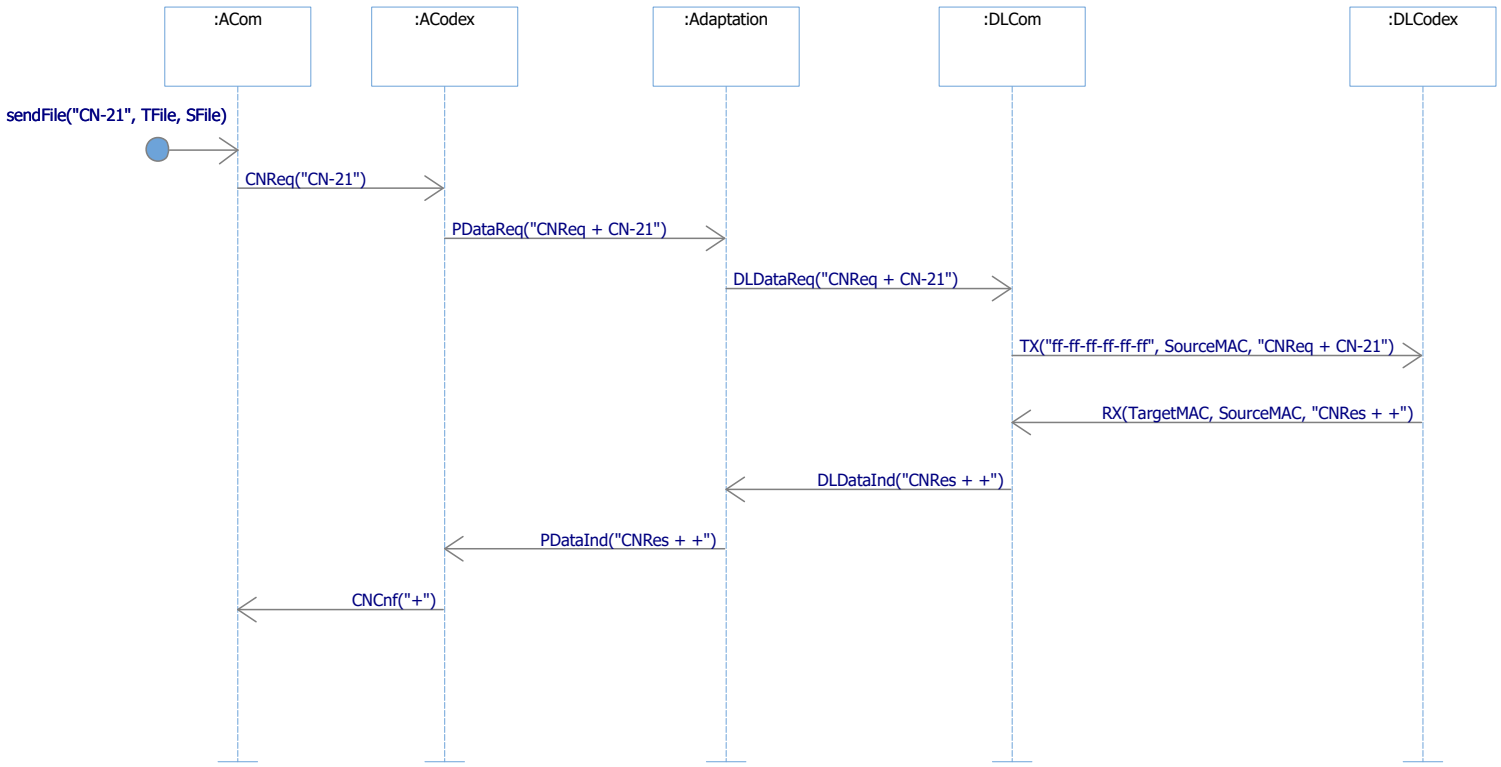
CNCnf schlaegt timeout

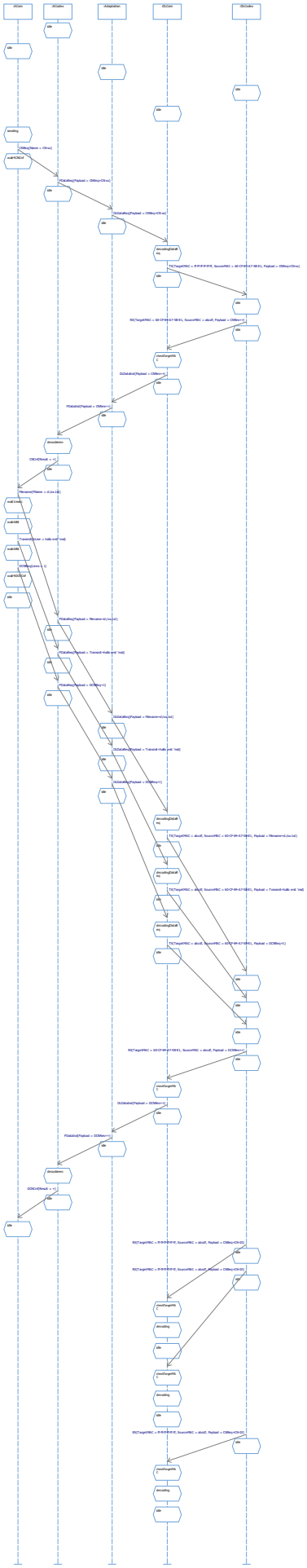


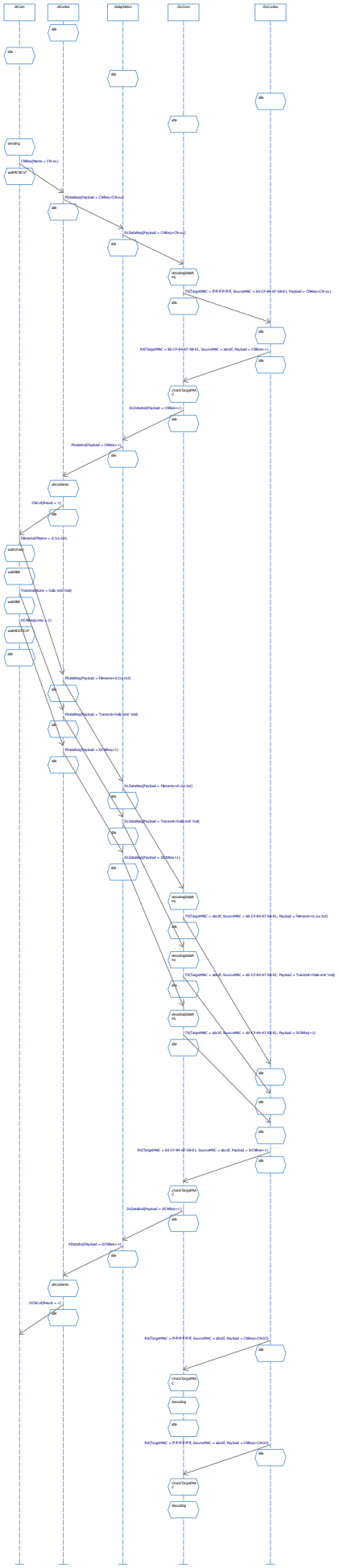
DCNCnf schlaegt timeout



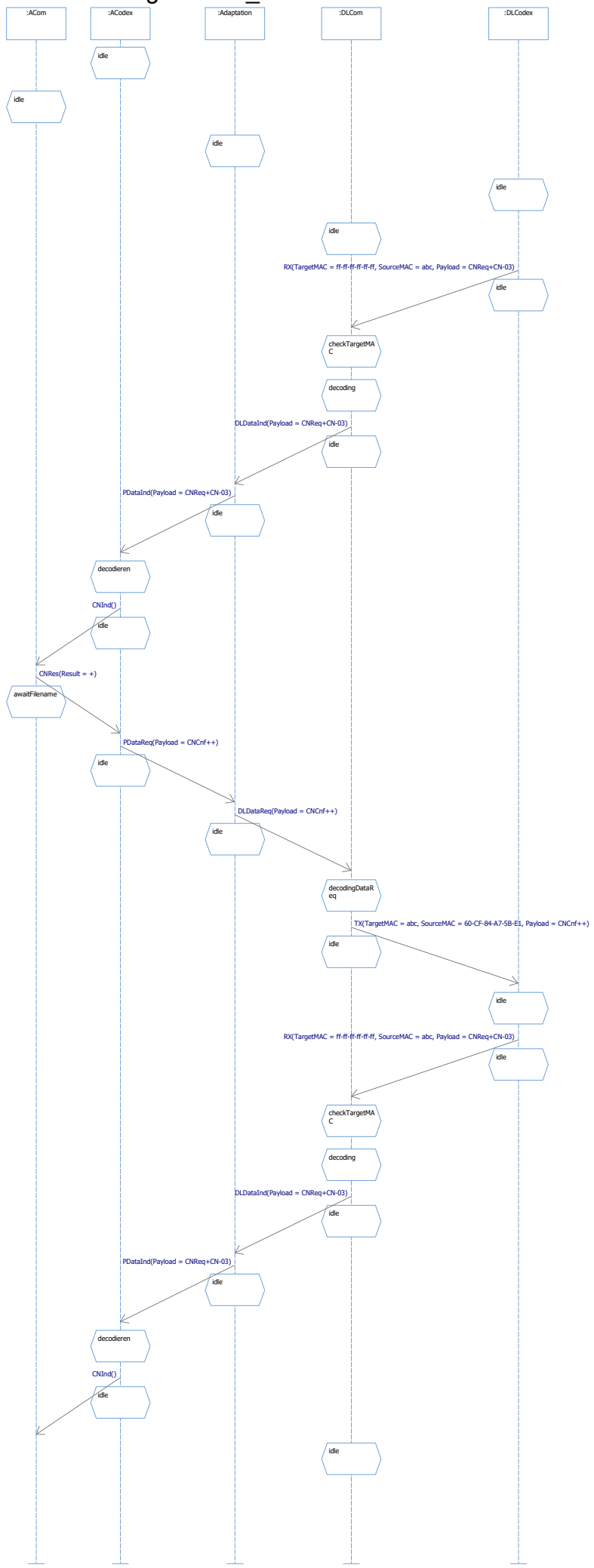
Verbindungsaufbau

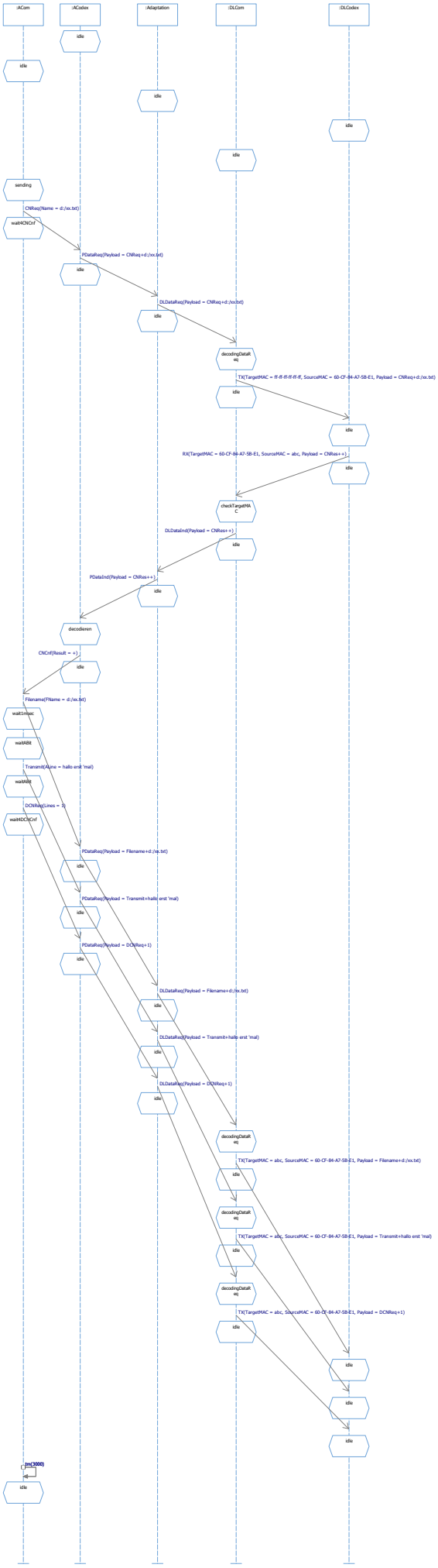



JCCom
JCCodec
Adaptation
DLCom
DLCodec
idle
sending
CMReq(Name = CHrxc)
wait4CNCnf
FDataReq(Payload = CMReq=CHrxc)
idle
DLDataReq(Payload = CMReq=CHrxc)
idle
decoding(Data Rx)
TX(Target/MAC = ff-ff-ff-ff-ff-ff, Source/MAC = 60-CF-8F-A7-5B-E1, Payload = CMReq=CHrxc)
idle
idle
RX(Target/MAC = 60-CF-8F-A7-5B-E1, Source/MAC = abcdf, Payload = CMResp=1)
idle
checkTargetMAC
DLDataInd(Payload = CMResp=1)
idle
FDataInd(Payload = CMResp=1)
idle
decoderesp
CNCnf(Result = 1)
idle
Filename(FName = d.txt,16)
wait1MB
wait4MB
Transmit(Size = hallo end "end)
wait1MB
DCMReq(Lines = 1)
wait4DCCnf
idle
FDataReq(Payload = Filename=d.txt,16)
idle
FDataReq(Payload = Transmit=hallo end "end)
idle
FDataReq(Payload = DCMReq=1)
idle
DLDataReq(Payload = Filename=d.txt,16)
idle
DLDataReq(Payload = Transmit=hallo end "end)
idle
DLDataReq(Payload = DCMReq=1)
idle
decoding(Data Rx)
TX(Target/MAC = abcdf, Source/MAC = 60-CF-8F-A7-5B-E1, Payload = Filename=d.txt,16)
idle
decoding(Data Rx)
TX(Target/MAC = abcdf, Source/MAC = 60-CF-8F-A7-5B-E1, Payload = Transmit=hallo end "end)
idle
decoding(Data Rx)
TX(Target/MAC = abcdf, Source/MAC = 60-CF-8F-A7-5B-E1, Payload = DCMReq=1)
idle
idle
idle
idle
RX(Target/MAC = 60-CF-8F-A7-5B-E1, Source/MAC = abcdf, Payload = DCMResp=1)
idle
checkTargetMAC
DLDataInd(Payload = DCMResp=1)
idle
FDataInd(Payload = DCMResp=1)
idle
decoderesp
DCMCnf(Result = 1)
idle
idle
RX(Target/MAC = ff-ff-ff-ff-ff-ff, Source/MAC = abcdf, Payload = CMReq=CHrxc)
idle
RX(Target/MAC = ff-ff-ff-ff-ff-ff, Source/MAC = abcdf, Payload = CMReq=CHrxc)
idle
checkTargetMAC
decoding
idle
checkTargetMAC
decoding
idle
RX(Target/MAC = ff-ff-ff-ff-ff-ff, Source/MAC = abcdf, Payload = CMReq=CHrxc)
idle
checkTargetMAC
decoding
idle


JCCom
JCCodec
JAdaptation
DLCom
DLCodec
idle
idle
idle
idle
idle
sending
CMReq(Name = Ch-xx)
waitHCNCnf
PDataReq(Payload = CMReq+Ch-xx)
idle
DLDataReq(Payload = CMReq+Ch-xx)
idle
decodingdatafr
ame
TX(TargetMAC = ff-ff-ff-ff-ff-ff, SourceMAC = 60-CF-84-A7-5B-E1, Payload = CMReq+Ch-xx)
idle
idle
RX(TargetMAC = 60-CF-84-A7-5B-E1, SourceMAC = abcdf, Payload = CMAck+1)
idle
checkTargetMA
C
DLDataInd(Payload = CMAck+1)
idle
PDataInd(Payload = CMAck+1)
idle
decoderen
CNCnf(Result = 1)
idle
Filename(Name = d.txx.txt)
waitConnect
waitAck
Transmit(Name = talk-ent'mal)
waitAck
DCMReq(Axx = 1)
waitHCNCnf
idle
PDataReq(Payload = Filename=d.txx.txt)
idle
PDataReq(Payload = Transmit+talk-ent'mal)
idle
PDataReq(Payload = DCMReq+1)
idle
DLDataReq(Payload = Filename=d.txx.txt)
idle
DLDataReq(Payload = Transmit+talk-ent'mal)
idle
DLDataReq(Payload = DCMReq+1)
idle
decodingdatafr
ame
TX(TargetMAC = abcdf, SourceMAC = 60-CF-84-A7-5B-E1, Payload = Filename=d.txx.txt)
idle
decodingdatafr
ame
TX(TargetMAC = abcdf, SourceMAC = 60-CF-84-A7-5B-E1, Payload = Transmit+talk-ent'mal)
idle
decodingdatafr
ame
TX(TargetMAC = abcdf, SourceMAC = 60-CF-84-A7-5B-E1, Payload = DCMReq+1)
idle
idle
idle
idle
RX(TargetMAC = 60-CF-84-A7-5B-E1, SourceMAC = abcdf, Payload = DCMAck+1)
idle
checkTargetMA
C
DLDataInd(Payload = DCMAck+1)
idle
PDataInd(Payload = DCMAck+1)
idle
decoderen
DCNCnf(Result = 1)
idle
RX(TargetMAC = ff-ff-ff-ff-ff-ff, SourceMAC = abcdf, Payload = CMReq+Ch-01)
idle
checkTargetMA
C
decoding
idle
RX(TargetMAC = ff-ff-ff-ff-ff-ff, SourceMAC = abcdf, Payload = CMReq+Ch-02)
idle
checkTargetMA
C
decoding

Animated Verbindungsaufbau_2




:ACom
:ACodex
:Adaptation
:DLCom
:DLCodex
idle
sending
CNReq(Name = d:/xx.txt)
wait4CNCnf
PDataReq(Payload = CNReq+d:/xx.txt)
DLDataReq(Payload = CNReq+d:/xx.txt)
decodingDataReq
TX(TargetMAC = ff-ff-ff-ff-ff-ff, SourceMAC = 60-CF-84-A7-5B-E1, Payload = CNReq+d:/xx.txt)
RX(TargetMAC = 60-CF-84-A7-5B-E1, SourceMAC = abc, Payload = CNRes++)
checkTargetMAC
DLDataInd(Payload = CNRes++)
PDataInd(Payload = CNRes++)
decodieren
CNCnf(Result = +)
Filename(FName = d:/xx.txt)
wait1msec
wait8Bit
Transmit(Line = hallo erst 'mal)
DCNReq(Lines = 3)
wait4DCNCnf
PDataReq(Payload = Filename+d:/xx.txt)
PDataReq(Payload = Transmit+hallo erst 'mal)
PDataReq(Payload = DCNReq+1)
DLDataReq(Payload = Filename+d:/xx.txt)
DLDataReq(Payload = Transmit+hallo erst 'mal)
DLDataReq(Payload = DCNReq+1)
TX(TargetMAC = abc, SourceMAC = 60-CF-84-A7-5B-E1, Payload = Filename+d:/xx.txt)
TX(TargetMAC = abc, SourceMAC = 60-CF-84-A7-5B-E1, Payload = Transmit+hallo erst 'mal)
TX(TargetMAC = abc, SourceMAC = 60-CF-84-A7-5B-E1, Payload = DCNReq+1)
tm(3000)