Hochschule Mittweida (FH)
University of Applied Sciences

DIPLOMARBEIT

Markus Franke

Modellierung, Konzeption und Realisierung eines CSTA - Treibers

2010

Hochschule Mittweida (FH)
University of Applied Sciences
Fachbereich Informationstechnik & Elektrotechnik

DIPLOMARBEIT

Modellierung, Konzeption und Realisierung eines CSTA -Treibers

Autor: **Markus Franke**

Studiengang: Elektrotechnik/Kommunikationstechnik
Seminargruppe: ET05wK1

Erstprüfer: Prof. Dr. rer. nat. Sergej Alekseev
Zweitprüfer: Dipl.-Ing. Rico Thomanek

Mittweida, Mai 2010

Bibliografische Angaben

Franke,Markus:
Modellierung, Konzeption und Realisierung eines CSTA - Treibers
137 Seiten, 42 Abbildungen, 1 Anlage mit Quelltexten,
Hochschule Mittweida (FH), Fachbereich Informationstechnik & Elektrotechnik
Diplomarbeit, 2010

Referat:

Das Computer Supported Telecommunication Application (CSTA) Protokoll , gehört zur Computer Supported Integration (CTI), und verbindet das Telekommunikationsnetzes mit dem Computernetzwerk.
Diese Diplomarbeit befasst sich mit der Umsetzung eines CSTA Treibers. Mit dem Ziel diesen in das vorhandene Zeiterfassungssystem der Hochschule Mittweida zu integrieren. Zum Realisieren und Testen stand eine Siemens HiPath4000 zur Verfügung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CSTA	Computer Supported Telecommunication Application
ASN.1	Abstract Syntax Notation One
CTI	Computer Integrated Telephony
IEC	International Electronic Commission
ITU-T	International Telecommunication Union- Telecommunication Sector
ISO	International Organization for Standardization
ROSE	Remote Operations Service Element
PER	Packed Encoding Rules
CER	Cononical Encoding Rules
DER	Distinguished Encoding Rules
BER	Basic Encoding Rules
PDU	Protocol Data Unit

Vorwort

Die vorliegende Diplomarbeit wurde in der Forschungsgruppe „Telekommunikation“ in der Hochschule Mittweida (FH) angefertigt.

Ich möchte mich ausdrücklich bei all denen bedanken, die an der Entstehung dieser Arbeit mitgewirkt haben, insbesondere bei Herrn Prof. Dr. rer. nat. S. Alekseev und Herrn M. Sc. R. Thomanek für die Durchsicht der Diplomarbeit und für die Unterstützung während der Bearbeitungszeit.

Außerdem möchte ich mich bei allen Mitarbeitern bedanken.

1 Einleitung

1.1 Motivation

In allen Betrieben, die einen großen Kundenstamm verwalten, ist es notwendig, schnell und zuverlässig Kundendaten zu speichern und abzurufen. Für den Fall, dass ein Kunde im Unternehmen anruft, wäre es sehr vom Vorteil, anhand der übermittelten Telefonnummer auf den Kundennamen zu schließen und dem Bearbeiter am Arbeitsplatz alle notwendigen Kundendaten automatisch anzeigen zu lassen. Wie aus diesem Beispiel ersichtlich ist, wird durch eine Vorverarbeitung der Informationen bei eingehenden Anrufen die Effizienz gesteigert und damit Kosten gesenkt. Um eine solche Vorverarbeitung zu ermöglichen, ist ein Computersystem mit einer Datenbank notwendig. In dieser Datenbank befinden sich alle Informationen über die Kunden und deren Telefonnummer. Die Bedeutsamkeit wurde frühzeitig von den Herstellern erkannt und es entstanden viele unterschiedliche Schnittstellen, damit beim Wechsel der PBX Hersteller nicht auch noch das System auf eine neue Schnittstelle angepasst werden muss. Daher entwickelte der ITU – T, ISO und IEC den Standard CSTA. Durch die Verbindung dieser beiden Welten lässt sich eine hohe Anzahl an weiteren Diensten realisieren.
Ein Mögliches Szenario wäre der Einsatz eines Zeiterfassungssystems. Dabei ließe sich in einer Datenbank die Arbeitszeiten der Arbeitnehmer sichern. Das Eintragen des Arbeitsbeginns und Arbeitsendes ließe sich durch die Telefone am Arbeitsplatz realisieren. Diese würden die Funktion eines Terminals einnehmen.

1.2 Kapitelübersicht

Das **Kapitel 1** soll in das Thema der Diplomarbeit einführen und einen kurzen Überblick geben, welche Inhalte in den einzelnen Kapiteln der Diplomarbeit abgehandelt werden.

Im **Kapitel 2** werden alle notwendigen Grundlagen für das Verständnis der Diplomarbeit vorgestellt. Zu Beginn wird die Abstrakt Syntax Notation Nummer 1 (kurz ASN.1) beschrieben. Darin wird nach der Beschreibung des Ursprungs auf die Encoding Rules und die verwendete Grammatik eingegangen.

Im Zweiten Unterpunkt von **Kapitel 2** wird der Standard CSTA behandelt und eine Übersicht über die Verwendung und Einsatzgebiete gegeben.

Zum Abschluss findet die Vorstellung von UML statt und dabei wird nur auf die relevanten Diagrammtypen der Diplomarbeit eingegangen.

Die Auswertung und Bewertung der Recherche findet im **Kapitel 3** statt. Darin werden die Auswahlkriterien für die ASN.1 Compiler beschrieben. Das Kapitel endet mit einem Fazit.

Im **Kapitel 4** und **Kapitel 5** ist der eigene Anteil an der Diplomarbeit beschrieben. **Kapitel 4** beschreibt die Konzeption und Modellierung des Treibers. Die Umsetzung und die dazugehörigen Test enthält **Kapitel 5**.

Die Diplomarbeit wird durch eine ausführliche Auswertung im **Kapitel 6** abgeschlossen.

1.3 Aufgabenstellung

Die Recherche steht am Anfang der Arbeit. Es gilt mögliche ASN.1 Compiler zu finden und diese zu bewerten. Nach Abschluss der Recherche steht die Modellierung und Konzeption an. Zur besseren Veranschaulichung wird der Treiber in seinen Grundlagen mittels der Beschreibungssprache UML skizziert. Nach Klären aller Randbedienung erfolgt die Umsetzung und Implementation des Treibers. Während und nach der Implementation des Treibers erfolgt die Testphase.

2 Grundlagen

2.1 Übersicht

In diesem Kapitel werden alle notwendigen Grundlagen bearbeitet**,** um die Thematik der Diplomarbeit verstehen zu können und die Lösungsansätze nachzuvollziehen.

Als erstes wird die Beschreibungssprache ASN.1 im **Kapitel 2.2** erläutert, dabei geht der Autor kurz auf den Ursprung und die Verwendung ein. Einen größeren Abschnitt nehmen die Grammatik und die Encoding Rules ein.

Im **Kapitel 2.3** wird das Interface CSTA beschrieben. Im **Kapitel 2.3.2** wird kurz auf den Einsatz und die Bedeutung eingegangen.

Im **Kapitel 2.4** wird die Modellierungssprache UML behandelt.

2.2 Einführung in ASN.1

ASN.1 [1]steht für Abstract Syntax Notation One und ist eine Sprache zur Beschreibung strukturierter Informationen. Es handelt sich um Informationen, die über Schnittstellen oder Kommunikationsmedien übertragen werden. Die Beschreibung der Nachricht erfolgt auf einem sehr hohen Abstraktionsniveau. ASN.1 ist international standardisiert und wird meistens dazu verwendet, um Kommunikationsprotokolle zu spezifizieren.

In der Nachfolgenden Grafik sollen einige Vorteile von ASN.1 dargestellt werden.

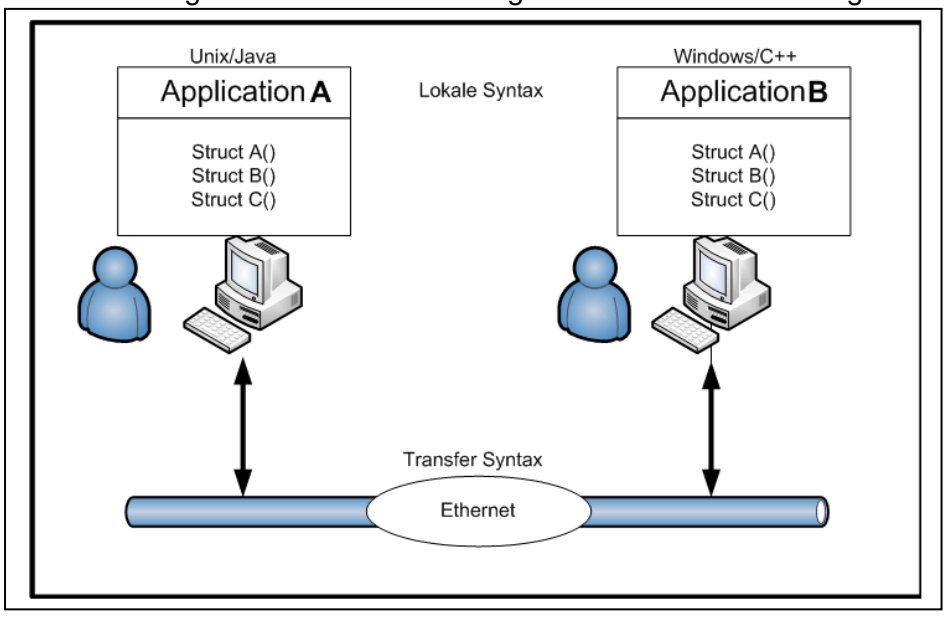


Abbildung 1: Anwendungsbeispiel für CSTA

[1] Abstract Syntax Notation One in X.680 standardisiert

Wie auf dieser Abbildung zu sehen sind zwei PC‘s über Ethernet miteinander verbunden. Auf dem ersten PC handelt es sich um ein UNIX Betriebssystem, worauf eine in JAVA programmierte Applikation A ausgeführt wird. Der Zweite PC arbeitet mit einem Windows Betriebssystem, worauf eine in C++ programmierte Applikation B ausgeführt wird. Damit die Applikationen A Informationen mit Applikation B austauschen kann. Wird die lokale Syntax von Applikation A in eine Transfer Syntax umgewandelt. Diese Umwandlung basiert auf Regeln der Encoding Rules. Auf der Seite von Applikation B wird diese empfangene Syntax wieder in eine lokale, für die Anwendung verständliche, Syntax umgewandelt.

Durch das Umwandeln der Syntax in eine für beide Systeme verständliche Form ist eine Kommunikation der Applikationen möglich.

2.2.1 Überblick

In diesen Unterkapiteln wird zum ersten der Ursprung beschrieben. Nachfolgend sollen einige Beispiele genannt werden, in welchem Bereich heutzutage ASN.1 Anwendung findet. Abschließend soll im Unterpunkt 2.2.4 die Grammatik und die Encoding Rules vorgestellt und an Beispielen erklärt werden.

2.2.2 Ursprung

Die Entwicklung von ASN.1 begann mit der rasanten Entwicklung der elektronischen Kommunikationssystemen und deren Vernetzung. Zur Realisierung der verschiedenen Dienste, wie z.B. E-Mail, Fax oder VOIP. Zur Realisierung dieser Dienste ist eine Reihenfolge von Meldungen notwendig, die zwischen den involvierten Zwischenstellen ausgetauscht werden müssen. Mit Hilfe von ASN.1 können alle Systeme auf einer gemeinsamen logischen Plattform kommunizieren.

Anfang der 80er Jahre entstand die Abstract Notation One.

Die Standardisierung wurde von der ITU - T [2], IEC[3] und ISO[4] vorangetrieben.

[2] International Telecommunication Union- Telecommunication Sector

[3] International Electronic Commission

[4] International Organization for Standardization

In der nachfolgenden Tabelle sind die beschrieben Standards chronologisch geordnet[1]:

Standard	Beschreibung
X. 680 (8824-1)	Spezifikation der Standard – Notation
X. 681 (8824-2)	Spezifikation von Informationsobjekten
X. 682 (8824-3)	Spezifikation von Untermengen (Constraints)
X. 683 (8824-4)	Parametrisierung von ASN.1 Spezifikationen
X. 690 (8825-1)	ASN.1 Codierungsregeln. Spezifikation von : - BER[5] - CER[6] - DER[7]
X. 691 (8825-2)	ASN.1 Codierungsregeln Spezifikation von PER[8]

Tabelle 1: Standards in denen ASN.1 beschrieben wird.

2.2.3 Verwendung

Der Einsatz von ASN.1 ist in jedem Gebiet sinnvoll, in welchem die Syntax von Datentypen und Datenwerten komplexer Informationen zu spezifizieren ist.
Für die bessere Verdeutlichung wird anhand der OSI-Schichten der Einsatz von ASN.1 gekennzeichnet.

[5] Basic Encoding Rules

[6] Canonical Encoding Rules

[7] Distinguished Encoding Rules

[8] Packed Encoding Rules

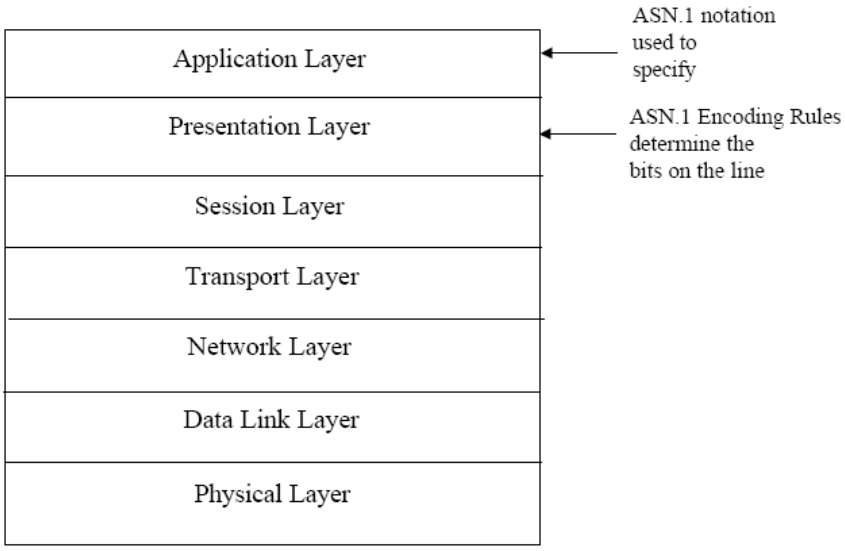


Abbildung 2: Protokollstack mit ASN.1 [5]

Die Aufgabe der Darstellungsschicht[9] ist die Darstellung und die korrekte Umsetzung von Daten, um einen syntaktisch korrekte Datenaustausch zu gewährleisten.
Wie aus der Abbildung ersichtlich ist, wird ASN.1 der Darstellungsschicht zugeordnet. Damit Daten mit anderen Applikationen ausgetauscht werden, nutzt die Darstellungsschicht eine für alle Systeme verständliches Datenformat, wie ASN.1 und agiert somit als Übersetzter.

Durch ASN.1 können PDU's[10] sämtlicher Netzsysteme wie ISDN, ATM, LAN, GSM und Internet aus sämtlichen OSI-Schichten abstrakt definiert und konkret codiert werden.
Den größten Vorteil von ASN.1 ist, dass unabhängig vom Betriebssystem und verwendeter Programmiersprache, alle auf einer einheitlichen logischen Ebene kommunizieren.
Nicht nur PDU's lassen sich mit dieser Beschreibungssprache definieren, sondern auch Dienste, Schnittstellen und Primitives.

Für diese Diplomarbeit ist die Verwendung von ASN.1 in der CTI[11] relevant, da der Standard CSTA[12] in ASN.1 Notation beschrieben ist. Dieser soll dann in Form eines Treibers bzw. einer Schnittstelle umgesetzt werden.

[9] engl. Presentation Layer

[10] PDU – Protocol Data Unit

[11] CTI – Computer Telephony Integration

[12] CSTA – Computer Supported Telecommunications Applications von der ECMA standardisiert

2.2.4 ASN.1 Syntax und Grammatik

Zu Beginn soll die ASN.1 Grammatik und anschließend der Aufbau ausgewählter Encoding Rules erklärt werden.

Syntax

Anhand des nachfolgenden Beispiels „Make Call“ soll die grundlegende Syntax erklärt werden. Auf die genauere Bedeutung geht der Autor erst im späteren Verlauf der Arbeit ein.

```
1.    CSTA-make-call
2.    { iso( 1) identified-organization( 3) icd-ecma( 12)
3.    standard( 0) csta3( 285) make-call( 10) }
4.    DEFINITIONS ::=
5.    BEGIN
6.    IMPORTS
7.    DeviceID FROM CSTA-device-identifiers
8.    { iso( 1) identified-organization( 3) icd-ecma( 12)
9.    standard( 0) csta3( 285) device-identifiers( 123) };

10.   makeCall    OPERATION ::= {
11.         ARGUMENT    MakeCallArgument
12.         RESULT            MakeCallResult
13.         ERRORS            {universalFailure}
14.         CODE        local: 10
            }

15.   MakeCallArgument ::=    SEQUENCE {
16.         callingDevice           DeviceID,
17.         calledDirectoryNumber         DeviceID
             }

18.   MakeCallResult ::= SEQUENCE {
19.         callingDevice           ConnectionID
            }
20.   END  -- of CSTA-make-call
```

Die ASN.1 Beschreibung wird in BNF[13] angeben.

[13] Backus-Naur-Form

Das Grundelement der ASN.1 ist das Modul (Zeile 1 -23). Der Zweck eines Moduls ist die Zusammenfassung und Bezeichnung einer Sammlung von Definitionen und/ oder Wert – Definitionen.

Die Definition eines Moduls beginnt immer mit dem Modul-Namen, gefolgt von einem optionalen Object-Identitifer. Danach folgen das Schlüsselwort **DEFINITIONS** und der Zuweisungs-Zeichenfolge „**:: =**“. Der Modulkörper wird mit dem Schlüsselwort **END** abgeschlossen.

```
Modul-name {1} DEFINITIONS ::=
BEGIN
IMPORTS
Name-der-einzubindenden-Struktur
{object Identifier};

      Definitionen ::= ...
END
```

Mit Hilfe des Schlüsselwortes **IMPORTS** können Datentypen importiert werden, dadurch spart man einen erheblichen Mehraufwand ein.

Object Identifier verweist auf die Adresse des benötigten Datentyps.

Da es bei mehreren Strukturen mit eigenen Datentypen schnell unübersichtlich werden kann, wird eine Modularisierung bevorzugt. Dabei werden einzelne Funktionen bzw. Strukturen zu Modulen zusammengefasst. In der Programmiersprache C ist dies mit dem Einbinden von Header-Files vergleichbar.

Ein Modul beginnt immer mit dem Modul-Namen. Dieser bezeichnet meist den Inhalt des Moduls.

Nachfolgend wird jedem Modul eine weltweit einzigartige Kennung, dem sogenannten „module identifier“, zugeordnet.

Mit Hilfe der Struktur „Make-Call“ aus dem CSTA –Standard soll die Funktionsweise von IMPORTS erklärt werden.

```
            ...
6.    IMPORTS

7.    DeviceID FROM CSTA-device-identifiers
8.    { iso( 1) identified-organization( 3) icd-ecma( 12)
9.    standard( 0) csta3( 285) device-identifiers( 123) };

            …

15.   MakeCallArgument ::=    SEQUENCE {
16.         callingDevice                    DeviceID,
17.         calledDirectoryNumber        DeviceID
             }
            ...
```

Anhand dieses Beispiels ist zu erkennen, dass sowohl “CallingDevice“ als auch „calledDirectoryNumber“ vom Typ „DeviceID“ ist. Da mehrere Strukturen auf diesen eigenen Datentyp zugreifen, ist es sinnvoll diese Datentypen in einer eigenen Struktur zu definieren. Wenn andere Datenstrukturen diesen Datentypen verwenden, kann dies durch die Funktion IMPORTS eingebunden werden. Der Object Identifier verweist dann auf die Adresse des benötigten Datentyps. In Zeile 7 findet das Einbinden des unbekannten Typs statt, detailliert dargestellt in folgender Abbildung.

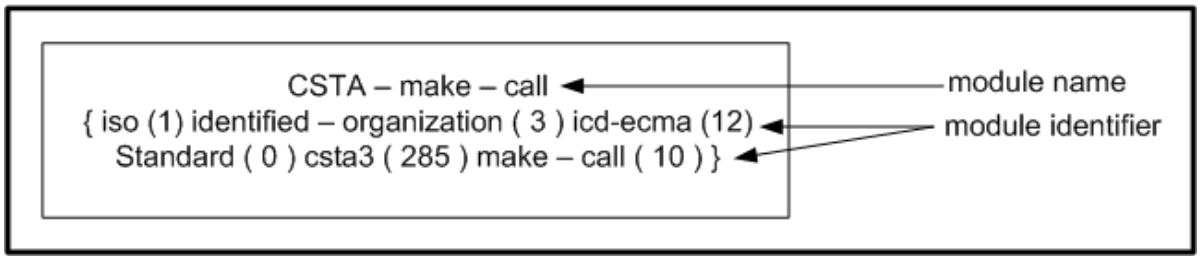


Abbildung 3: Modulheader

„Module name“ und „module identifier“ bilden den Modulheader.

Der OID wird von der Management Information Base (MIB) verwaltet. Die MIB ist eine Datenbank, die alle relevanten Daten in relationaler und objektorientierter Form für Management Systeme enthält.

Für jedes dieser Objekte gibt es eine eindeutige Bezeichnung den sogenannten Object Identifier. Der IOD kann sowohl als eine Zeichenkette (.1.3.6.1.4.1) als auch durch einen ASCII – String repräsentiert werden (.iso.org.dod.internet.private.enterprise).

Eine Grafische Darstellungsform bildet der Object Identifier Tree nachfolgend dargestellt.

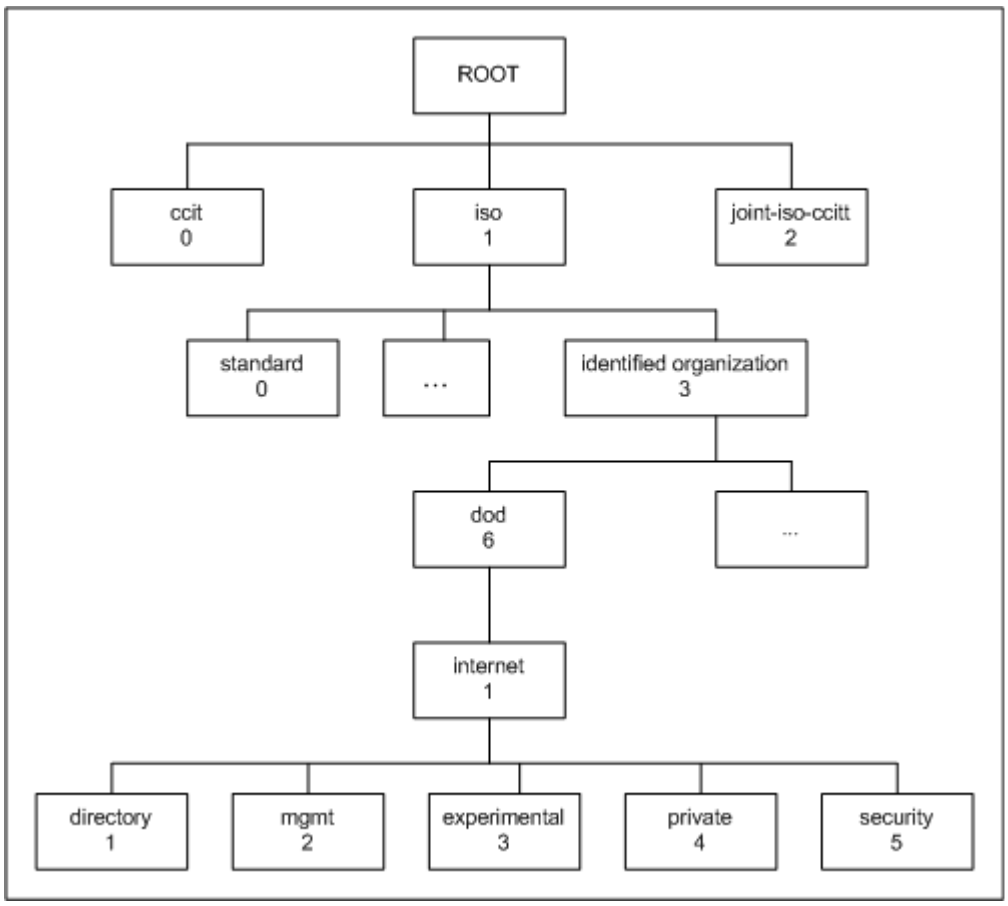


Abbildung 4:Object Identifier Tree

Grundlegende Grammatik

ASN.1 ähnelt in gewisser Weise Programmiersprachen, wie C oder Pascal. Nachfolgend stellt die Tabelle die Parallelität zwischen ASN.1 und C Typen dar.

ASN.1 Typen	C Typen
BOOLEAN	bool
INTEGER	int
ENUMERATED	Enum
REAL	Float
BIT STRING	BITS
OCTET STRING	OCTS
CHARACTER STRING	OCTS

Tabelle 2 : Vergleich ASN1 Typen mit C Typen

Die Syntax für Strukturen in C sind dem in ASN.1 sehr ähnlich. Zur Veranschaulichung soll eine Struktur in C und in ASN.1 verglichen werden.

Als Beispiel wird eine Struktur mit dem Namen „meineAdresse“ angelegt. Diese Struktur besteht aus name, vorname, strasse und hausnummer.

C	ASN.1

`Struct meineAdresse{`
 	
`char name [20]`
`char vorname [20]`
`char strasse [30]`
`int hausnummer`
`}`
	`DEFINITION ::=`
	`BEGIN`
	`meineAdresse ::= SEQUENCE {`
	
	`name PRINTABLESTRING (SIZE[0..20])`
	`vorname PRINTABLESTRING (SIZE[0..20])`
	`strasse PRINTABLESTRING (SIZE[0..30])`
	`hausnummer integer`
	`}`
	`END`

Tabelle 3: Vergleich der Programmiersprache C und ASN.1

Die Definierung erfolgt mit dem Zuweisungszeichen “::=”.

Den Parameternamen name, vorname und strasse wird der Datentyp PRINTABLESTRING zugewiesen. Mit dem Wort SIZE wird die maximale Anzahl der verwendeten Buchstaben festgelegt.

Wichtig für das Umwandeln der lokalen Syntax in die Transfer Syntax sind die sogenannten Tag-Klassen.

Es gibt vier Tag-Klassen[8]:

UNIVERSAL: Tags sind für Datentypen, welche im ASN.1 Standard vordefiniert sind, reserviert und besitzen universelle Gültigkeit. Hierbei wird wiederum in zwei verschiedene Arten von Datentypen, die atomic-Type und in die structure-Type unterteilt.

Als atomic-Types werden Datentypen bezeichnet, wie Boolean, Integer oder Real.

Wird ein Datentyp aus mehreren atomic-Types zusammengesetzt,

gehört dieser zu den structure-Types.

Um zusammengesetzte Typen zu verwenden und somit die

Programmierbarkeit zu erleichtern, ist es notwendig, eine Struktur zu definieren. Ähnlich wie in C, kann ebenfalls in ASN.1 mittels dem „structure“-Type Datenstrukturen angelegt werden.

APPLICATION: Tags ist nur für Anwendungen gültig. Für nähere Bedeutung muss der Standard heran gezogen werden.

PRIVATE: kann für eigene Spezifikationen verwendet werden

Context specific: besitzt keinerlei Einschränkungen.

Der Einsatz der einzelnen Tag-Klassen ist sehr von der Art des zu umsetzenden Projektes abhängig. Zur Vereinfachung soll nachfolgend nur auf die Universal Tag-Klasse eingegangen werden.

„ASN.1 ermöglicht es, neue zusammengesetzte Typen, also structured-types zu spezifizieren, die möglichen Mechanismen sind nachfolgender Tabelle aufgeführt.“ [8]

SEQUENCE	Folge von Komponenten, die verschiedene Typen besitzen, Reihenfolge muss beachtet werden.
SEQUENCE OF	Folge von Typen, die alle den gleichen Typen besitzen, Reihenfolge muss beachtet werden.
SET	Folge von mehreren Komponenten, die verschiedenen Typen besitzen, Reihenfolge wird nicht beachtet.
SET OF	Folge von Typen, die alle den gleichen Typ besitzen, Reihenfolge wird nicht beachtet.
CHOICE	Auswahl eines Elementes aus einer Liste

Tabelle 4 Datenstrukturen

Mit diesen Voraussetzungen können jetzt komplexe Strukturen in ASN.1 Syntax beschrieben und verstanden werden.

2.2.5 Encoding Rules

Die Encoding Rules übernehmen die Rolle, die beladene Struktur in einen Oktettstrom umzuwandeln. Diese Nachricht kann dann auf drahtgebundener oder Drahtungebundener Kommunikationswegen übertragen werden.

ASN.1 überdeckt ein breites Anwendungsgebiet. Einige davon stellen besondere Anforderungen an den Standard, wie z.B. geringe Bandbreite. Um diese Bedingungen zu erfüllen, wurde der Standard weiterentwickelt. Die Grundform stellt die Basic Encoding Rule (BER) dar.

Nachfolgend soll die Grundlagen der Encoding Rules bearbeitet werden.

Da es in der Realisierung der Diplomarbeit nur die BER-Codierung Anwendung findet, wird darauffolgend genauer eingegangen und die anderen Codierungsvorschriften nur am Rande erwähnt.

2.2.5.1 Basic Encoding Rules (BER)

2.2.5.1.1 Einordnung

Die Basic Encoding Rules ist die ursprüngliche Codierungsvorschrift, die im ASN.1 Standard ITU-T X.690 definiert ist. Sie beinhaltet grundlegende Codierungsregeln und wird benutzt, um ASN.1 Daten oder Typ-Strukturen in eine Transfersyntax zu überführen. Die Codierung in die Transfer Syntax erfolgt in einer ganzzahligen Anzahl von Oktetts.

Bei BER handelt es sich um eine Reihe von Regeln zur Codierung von ASN.1-Daten in einen Okttetstrom, welche über eine Kommunikationsverbindung übertragen werden können.

In BER wird Folgendes definiert:

- Methoden für die Codierung von ASN.1 Werten,
- Regeln zur Festlegung, wann eine bestimmte Methode verwendet wird und
- das Format von bestimmten Oktetten in den Daten.

Bei ASN.1 handelt es sich um eine Programmiersprache, wie z. B. „C“, wohingegen es sich bei BER um einen Compiler für diese Sprache handelt. Compiler sind plattformspezifisch, viele komplexe Programmiersprachen hingegen nicht. ASN.1 ist die Sprache, in der ein Standard geschrieben wird. Daten, die von einem Programm erstellt werden und die dem Standard entsprechen, werden umgangssprachlich häufig als „ASN.1 Daten“ bezeichnet. ASN.1 kann erst dann verwendet werden, wenn die „ASN.1 Daten“ zu einem Oktettstrom codiert und diese am Ziel problemlos decodiert werden können.

Anwendung findet BER bei der Codierung von SNMP[14] - Protokolldaten und dem digitalen Signatur Standard PKCS[15]#7. Auch im Bereich der Telekommunikation über ISDN findet BER in der Codierung von Protokollinformationen seine Anwendung.

Allerdings ist die Codierung mit BER durch verschiedene Probleme gekennzeichnet. So setzt die Codierung ausreichend hohe Übertragungsrate durch die nicht gerade platzsparende Codierungslänge voraus.

Außerdem wird BER im Bezug auf anwendungsspezifische Aufgaben als unflexibel angesehen.

[14] Simple Network Management Protocol

[15] Public Key Cryptographie

Aufbau

Die BER-Codierung wird durch das 3er Tupel B = {T, L, V} definiert.

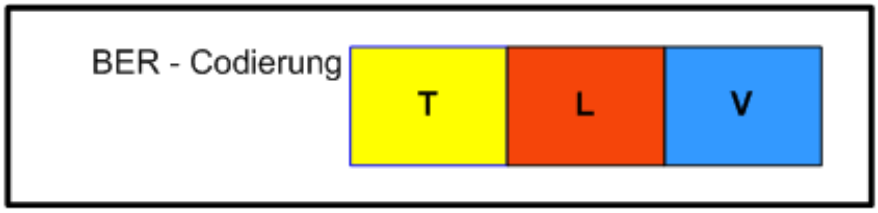


Abbildung 5: Aufbau Kodierung BER

B… Oktetts

T… Type gibt die Objekt-Klasse an

L… Length gibt die Anzahl der Bytes an, mit dem der Wert codiert wurde

V… Value enthält den eigentlichen übermittelten Wert des Elements

2.2.5.1.2 Data Types

Das erste Oktett in der BER Codierung gibt darüber Informationen, welche Struktur und welchen Aufbau die Nachricht besitzt und wie sie zu verarbeiten ist.
Die ersten beiden höchst wertigsten Bits gibt die Art der Tag-Klasse an.

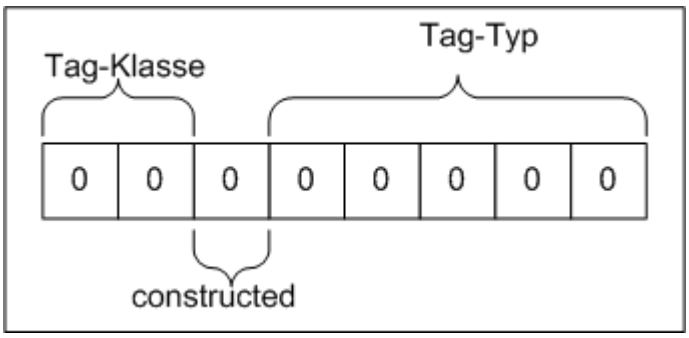


Abbildung 6: Tag Kodierung

Mit der Tag-Klasse lassen sich folgende vier Varianten darstellen.

Tag	Typ	Beschreibung
00	Universal	Im ASN.1 Standard definiert und besitzen universelle Gültigkeit
01	Application	für eine Anwendung gültig
10	Context – specific	besitzen keine Einschränkung, Kontext abhängig
11	Private	für eigene Spezifikation

Tabelle 5: Bedeutung der einzelnen Tag Klassen

Zur Angabe eines Constructed Datentyps wird das dritt höchst wertige Bit 1 gesetzt, andernfalls ist der Inhalt des Value Feldes vom Typ primitive.
Der Tag-Type gibt an, ob der Inhalt im Value Feld vom Typ Integer oder String usw. ist.
Benötigt das Tag-Type Feld mehr als 5 Bits, um den Inhalt zu encodieren, werden alle Bits auf „1“ gesetzt. Dadurch wird das nachfolgende Oktett auch zur Tag-Typ Encodierung genutzt.

2.2.5.1.3 Längencodierung

In der Längen Kodierung gibt es 3 Formen:

- Kurzform
- Langform
- Unbestimmte Form

Die **Kurzform** der Längencodierung besteht aus einem Oktett und hat einen Wertebereich von 1…127, dabei wird das höchst wertigste Bit auf 0 gesetzt und somit stehen 1 – 7 Bits zur Längencodierung zur Verfügung.

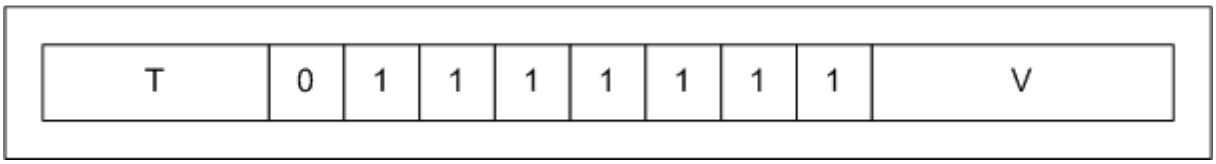


Abbildung 7: Längencodierung kurze Form

In der **Langform** wird das MSB 1 gesetzt und die darauf folgenden Bits geben die Anzahl der folgenden Oktetts, die zur Längencodierung verwendet werden, an.

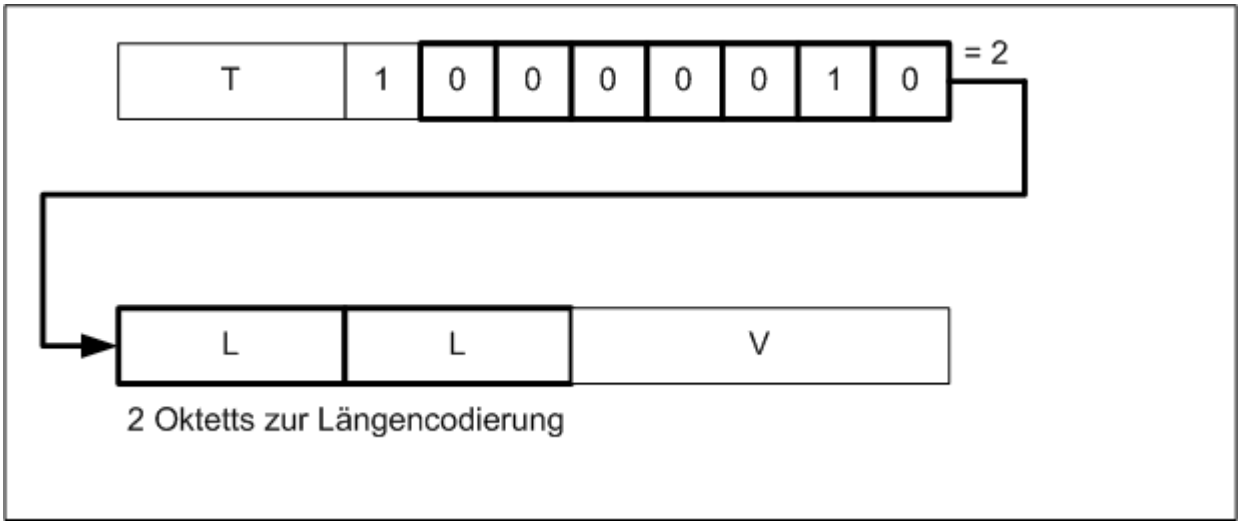


Abbildung 8:Längencodierung lange Form

Eine andere Möglichkeit, welche zum Einsatz kommt, wenn mehrere Oktetts zur Codierung benötigt werden, ist die **Unbestimmte Form**. In der Unbestimmten Form wird das MSB auf 1 gesetzt und alle anderen Bits im Oktett auf 0. Damit werden alle folgenden Oktette zur Längencodierung verwendet. Den Abschluss bilden zwei aufeinander folgende auf 0 gesetzte Oktetts.

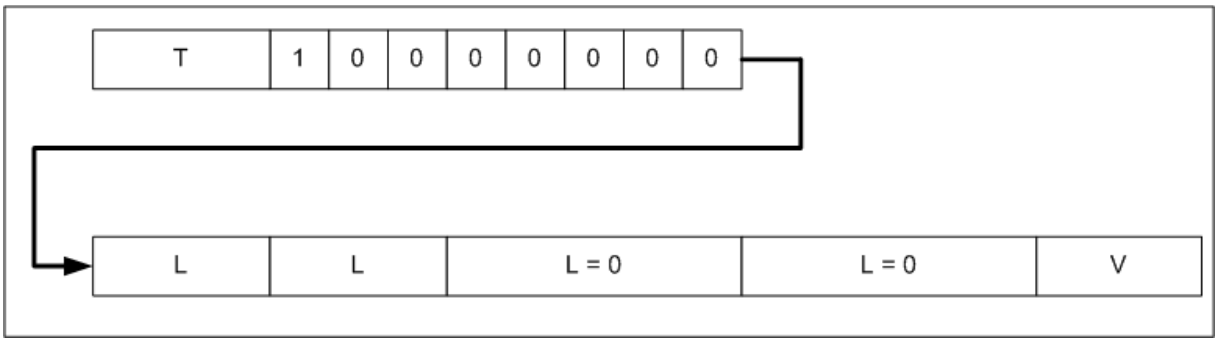


Abbildung 9: Längencodierung unbestimmte Form

2.2.5.1.4 Universal Tags

Bei der Umwandlung der Transfersyntax in die Abstrakte Syntax muss der verwendete Datentyp mitübertragen werden. Aus diesem Grund wurde jedem Datentyp ein eigener Tag-Wert zugewiesen. Eine Auswahl der wichtigsten Tag´s sind in der folgenden Tabelle aufgelistet.

Typ	Tag (dezimal)	Tag (Hex)
BOOLEAN	1	01
INTEGER	2	02
BIT STRING	3	03
OCTET STRING	4	04
REAL	9	09
ENUMERATED	10	0A
PRINTABLE STRING	19	13

Tabelle 6: Tabelle von Universal Tags

Wie in der oberen Tabelle zu sehen ist, würde bei einer Encodierung eines BOOLEAN Wertes der Typ-Tag 0x01 zugewiesen.

2.2.5.1.5 Beispiel

Zur besseren Verdeutlichung wird nachfolgend die Teststruktur „RectangleTest“ befüllt und nach den BER Encoding Rules codiert.

Zu Beginn die Teststruktur:

```
RectangleTest DEFINITIONS ::=
BEGIN
Rectangle ::= SEQUENCE {
height INTEGER,
width INTEGER
}
END
```

In diesem Bsp. wird „Height“ mit 15 und „Width“ mit 4 gefüllt.

Durch das Anwenden der Basic Encoding Rule (BER) wird die vorhandene Struktur in eine Bitfolge umgewandelt. Zur besseren Übersicht verwendet man das Hexadezimale Zahlensystem.

Als Oktettstrom erhält man „30 06 02 01 0F 02 01 04“.

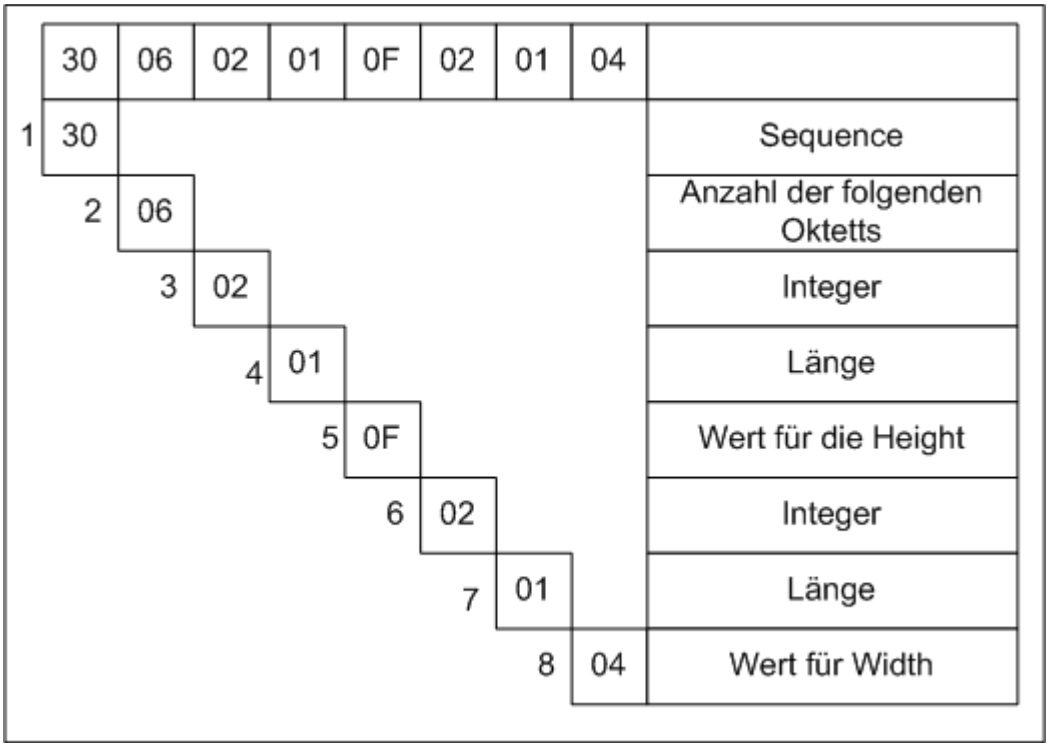


Abbildung 10: Beispiel Rectangle

Die Hexadezimale Zahl 0x30 steht für eine Sequence Datenstruktur. Der Nachfolgende Wert steht für die Anzahl der folgenden Bytes. In diesem Beispiel folgen 6 Bytes.

Der folgende Wert gibt an, von welchem Typ der Inhalt des Value Feld (Markierung 5) betrachtet wird.

Die Anzahl der verwendeten Bytes zur Encodierung des Value Feldes steht im Lenght Feld.

In beiden Fällen wird nur 1 Byte verwendet.

2.2.5.2 PER[16]

Bei dieser Art der Codierungsvorschrift erfolgt die Codierung nicht mehr in einer ganzzahligen Anzahl an Oktetts, sondern teilweise nur mit einzelnen Bits.

PER wird durch das 3er Tupel B = {P, L, C} definiert.

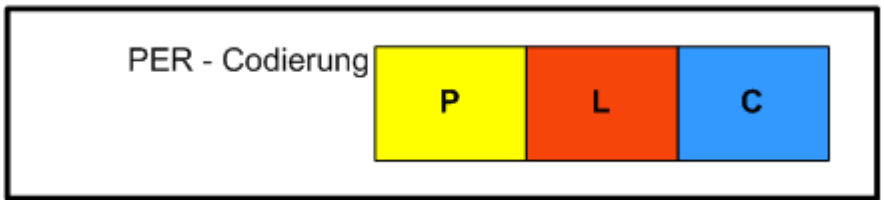


Abbildung 11: Aufbau PER - Codierung

P… Preamble - dieses Feld ist nur vorhanden, wenn in einer SEQUENCE optionale Elemente vorhanden sind bzw. in einer CHOICE angezeigt werden muss, welche Alternative gewählt wurden. Außerdem bestimmt es die Reihenfolge der Elemente eines SET's.

L… Length - Dieses Feld kann häufig ausgelassen werden, wenn die Länge im Voraus bekannt ist.

C… Content - Das Feld enthält den zu übermittelnden Inhalt des Elementes.

Da die Codierung mit PER platzsparend ist, findet es unter anderem Anwendungen in der Mobilfunktechnik.

Weitere Codierungsarten sind CER[17] und DER[18], die eine Untermenge der BER-Codierung darstellen.

Für die spezielle Anwendung mit XML Schemas bietet sich die XML Encoding Rule an, da nicht nur der Inhalt, sondern auch die Struktur übertragen wird.

[16] Packed Encoding Rules

[17] Canonical Encoding Rules

[18] Distinguished Encoding Rules

2.2.6 Nutzung eines ASN.1 Compilers

Die Notwendigkeit für den Einsatz eines ASN.1 Compiler ist überall da sinnvoll, wo es kompliziert zusammenhängende Strukturen umzusetzen gilt. Ein Vorteil die übersichtliche Erweiterungsmöglichkeit.

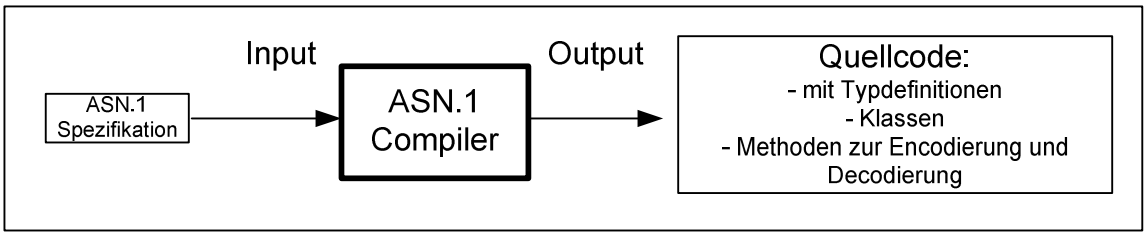


Abbildung 12: Einordnung des ASN.1 Compilers

Dem ASN.1 Compiler wird eine ASN.1 Spezifikation übergeben. Dieser erzeugt daraus Quellcode mit Typdefinitionen, Klassen und Compiler spezifische Methoden zur Encodierung und Decodierung.

Applikationen nutzen die erzeugten Methoden und eine Laufzeitbibliothek, um die in ASN.1 Syntax geschriebene Spezifikation umzusetzen.

Mit den erzeugten Files und einer Laufzeitbibliothek kann die Applikation den erzeugten Standard umsetzen.

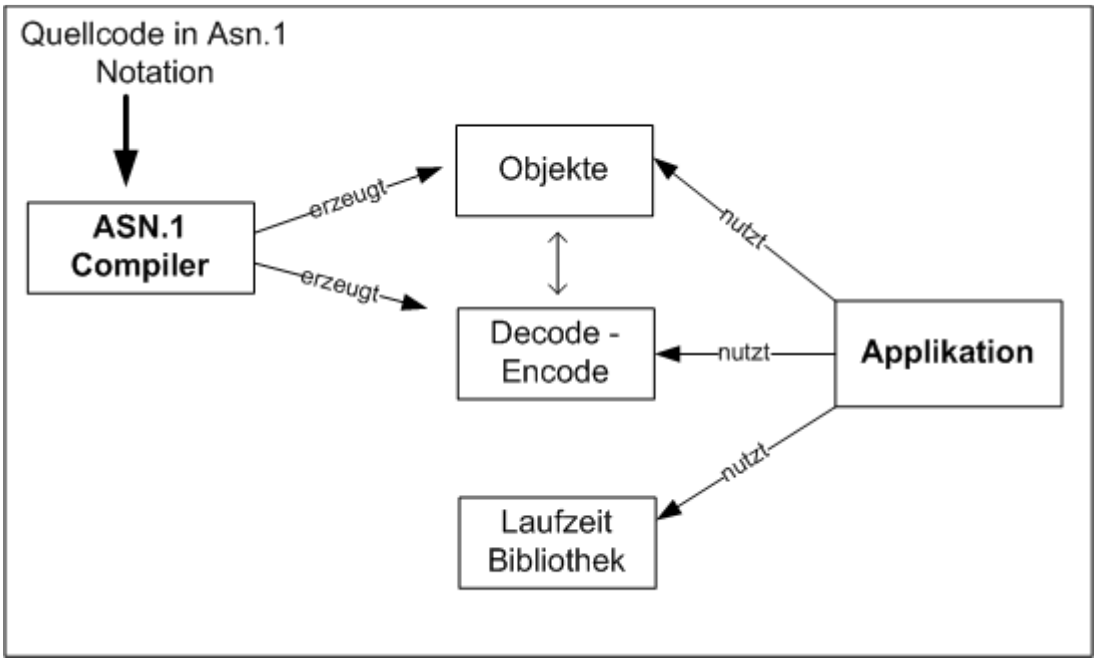


Abbildung 13: Verwendung eines ASN.1 Compilers

Ein guter ASN.1 Compiler erleichtert bei großen und unübersichtlichen Standards die Verwendung ungemein. Da aber gute ASN.1 Compiler auch sehr kostenintensiv sind, sollte stets geprüft werden, ob sich eine Investition rechnen würde.

2.3 Computer Supported Telecommunication Application

2.3.1 Überblick

CSTA ist ein Protokoll und wird dazu verwendet, um eine Kommunikation zwischen TK-Anlage und Rechner zu ermöglichen.

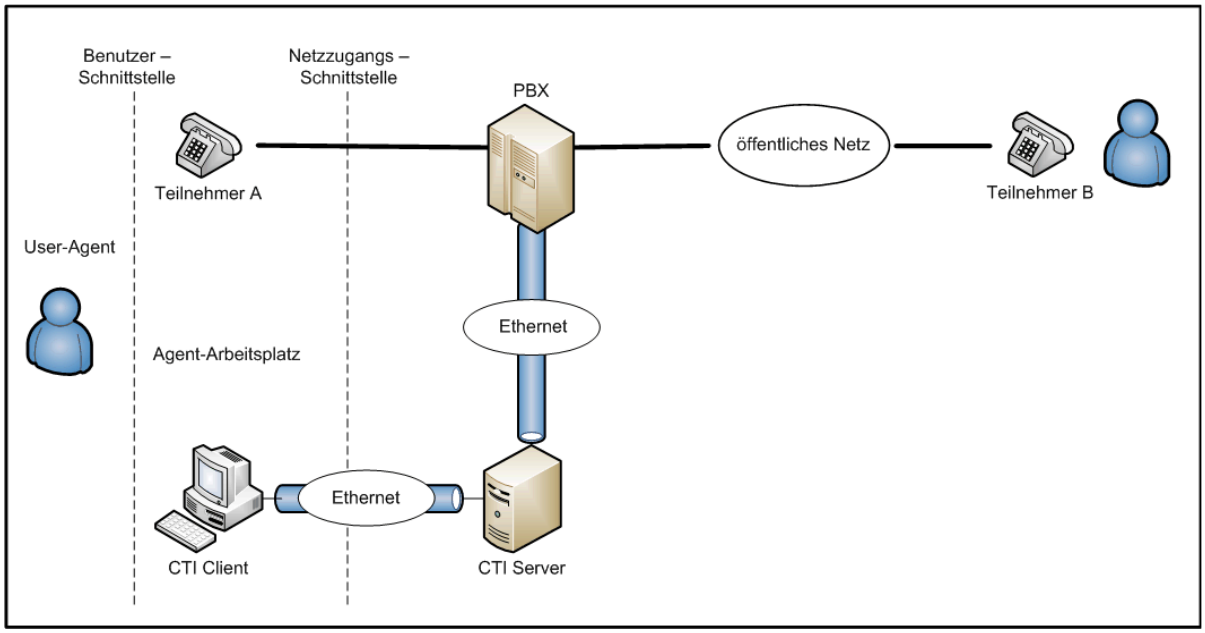


Abbildung 14: Einsatzszenario von CSTA

Wie in der oberen Grafik zu sehen, befindet sich auf der linken Seite der Benutzer. Ein Telefon ist mit einer PBX verbunden. Der CTI Client befindet sich im Ethernetnetzwerk mit dem CTI Server. Der CTI Server nutzt für die Kommunikation mit der PBX das TCP Protokoll. Der Teilnehmer B ist über ein öffentliches ISDN Netz mit der PBX verbunden.

Mit der CTI Client <---> PBX Verbindung wäre es möglich, ein Telefonat aufzubauen, welches vom CTI Client gesteuert wird.

2.3.2 Einsatz und Bedeutung

CTI verinnerlicht den Grundgedanken, Computer mit Telefonie zu verbinden, um ein Computer gestütztes Call Management zu realisieren. Dies ist vor allem für mittelständige Betriebe, welche einen hohen Anteil an Kundenservice besitzen, sinnvoll, da es dort zu einer erhöhten Telefonie aufkommen kann. Dem Benutzer stehen je nach Lösung unterschiedliche Funktionen zur Verfügung, wie Multicast, Makeln, Rückfragen, Konferenz, Telefonieren direkt aus der Datenbank bzw. den Kontakten heraus, Identifizierung eingehender Anrufe, Protokollierung aller Anrufe sowie Gruppenfunktionen wie Partnerliste, Instant Messaging oder Abwesenheitsnotiz zur Verfügung. Alle Funktionen werden dabei direkt vom PC aus gesteuert.

Die Standards für die Verbindung zwischen CTI-Lösung und TK-Anlage sind TAPI[19] und CSTA. Der Vorteil hierbei ist, dass man auch in Zukunft nicht von speziellen Herstellern abhängig sein wird.

CTI beinhaltet alle Werkzeuge und Methoden für die Zusammenarbeit von Computern und Telefonsystemen.

Dies hat einige Vorteile [6]:

- Interfaces zum Telefonnetz
- Audio Ein- und Ausgabefähigkeiten
- Vernetzung anderer Computernetze
- Allgemeine zeit- und kostengünstige Programmierbarkeit
- Bedienungsfreundlichkeit

[19] Telephony Application Programming Interface

CTI kann über zwei Architekturen realisiert werden:

First-Party-Lösung[7]	Telefonendgeräte sind über Serielle oder USB mit dem Rechner verbunden. Diese Lösung wird verwendet bei TK Anlagen die keine weiteren Schnittstellen besitzt oder nur wenige Arbeitsplätze mittels CTI–Schnittstellen ausgestattet werden. Die First-Party-Lösung wird hinsichtlich der Kopplung zur TK-Anlage unterteilt in phone centric und computer centric. Bei phone centric wird das Telefon direkt an der TK Anlage angeschlossen. Bei computer centric ist der Computer mit der TK Anlage angeschlossen.
Third-Party-Lösung	Die Telefonanlage ist direkt mit dem Server verbunden. Diese Lösung ist ratsam, wenn bereits ein gut ausgebautes Netzwerk besteht, außerdem lassen sich auf diesem Weg zusätzliche Dienste wie Instant Messaging und Abwesenheitsnotiz anbringen.

Tabelle 7: CSTA – Architekturen

Anwendungsbeispiele[3]:

- Telefonunterstützung
- Verbindungsaufbau
- Konferenzeinberufung
- Rückfrageeinteilung
- Systeminformationen abfragen
- Telemarketing
- Auf Basis Kundeninformationen Verbindungen vorgeschlagen / Kundenbetreuung
- Bei ankommenden Anrufen gleichzeitige Angabe von Kundendaten
- Datensammlung, Datenvermittlung
- Hotel: Zimmerservice, Zimmerstatus, Weckruf, Gebührenerfassung
- Krankenhaus: Patientenerfassung, Belegungsstatus, Notklingel

2.4 Unified Modeling Language

2.4.1 Grundlagen

UML ist eine Diagrammnotation zum Spezifizieren, Visualisieren und Dokumentieren von Modellen objektorientierter Softwaresysteme. Dies findet meistens Anwendung zur Pflege und Koordinierung größerer Softwareprojekte.

UML setzt sich aus vielen Modellelementen zusammen, die für sich einen bestimmten Sachverhalt des Softwaresystems repräsentieren. Diese Elemente werden zu Diagrammen kombiniert, die einen Ausschnitt oder einen bestimmten Blickpunkt auf das System darstellen.

Nachfolgend sind die wichtigsten Diagrammtypen aufgelistet. Insgesamt gibt es 13 Diagrammtypen.

- Anwendungsfalldiagramm (Use – Case Diagram)
- Klassendiagramm
- Sequenzdiagramm
- Kollaborationsdiagramm
- Zustandsdiagramm (State Diagram)
- Aktivitätsdiagramm
- Komponentendiagramm
- Verteilungsdiagramm
- Entitäten – Beziehungs-Diagramm

2.4.2 Vorstellung ausgewählter Diagrammtypen

In diesem Unterkapitel werden nur die Diagramme erklärt, die in dieser Diplomarbeit verwendet werden. Dazu gehören das Anwendungsfalldiagramm, Zustandsdiagramm und das Aktivitätsdiagramm.

2.4.2.1 Anwendungsfalldiagramm [9]

Ein Anwendungsfalldiagramm besteht aus einer Menge von Anwendungsfällen und stellt die Beziehung zwischen Akteuren und Anwendungsfällen dar. Es stellt das äußerlich erkennbare System aus der Sicht der Akteure dar.

Akteure können menschlich sein oder Partnersysteme, die Anwendungsfälle ausführen.

Darstellung und Beschreibung erfolgt durch folgende Symbole.

Durch Pfeile wird die Wirkrichtung angegeben.

- System
- Package
- Acteur
- Usecase

Die Beziehung von Akteuren und Anwendungsfällen oder Anwendungsfälle untereinander können durch Depency, Include, Extend, Generalzation, Directed Assoziation und Association beschrieben werden.

Zur besseren Veranschaulichung folgend ein Beispiel.

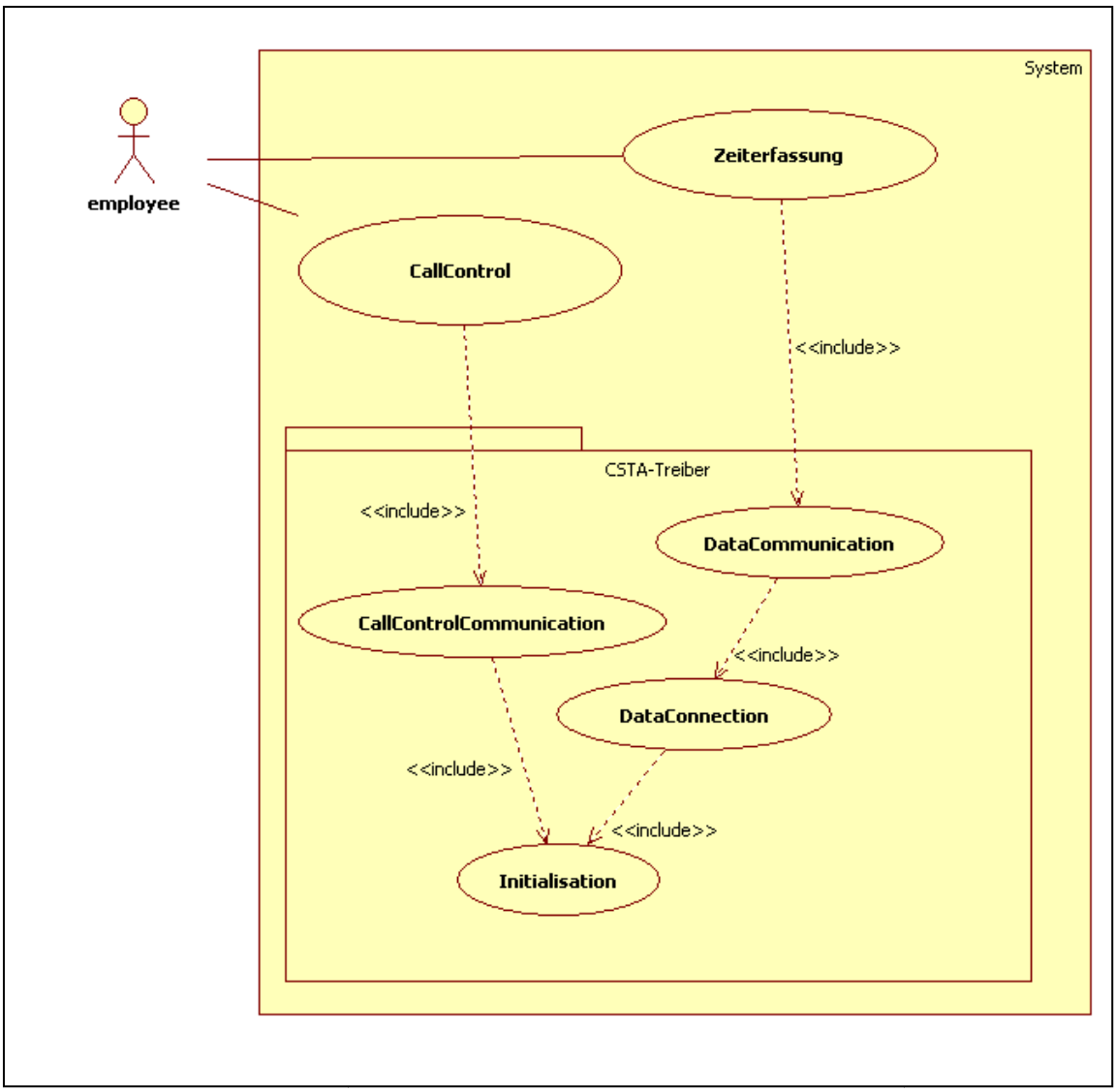


Abbildung 15: Beispiel eines Anwendungsfall Diagramms [4]

In diesem Beispiel ist der „employee“ der Akteur, der aktive Teil des Systems, der die „CallControl“- und die „Zeiterfassungs“-Dienste benutzt. Die Zeiterfassung ist mit der „DataCommunication“ des CSTA – Treibers assoziiert. Diese ist über „DataConnection“ mit „Initialisation“ verbunden. Dieser Diagrammtyp stellt nur Oberbegriffe dar, sagt aber noch nichts über ihre Implementation oder Funktionsweise aus.
Dieser Diagrammtyp hat den Vorteil, durch seine hohe Abstraktionsebene eine Übersicht über das gesamte Projekt zu geben. Deshalb sollte dieser der erste verwendete Diagrammtyp bei einem Softwareprojekt sein.

2.4.2.2 Zustandsdiagramm [10]

Ein Objekt kann während seiner Dauer verschiedenartige Zustände annehmen. Mit Hilfe des Zustandsdiagrammes kann man diese darstellen. Außerdem lassen sich auch die Aktionen, die zu einem Zustandsänderung führen würde, angeben.

Ein Zustandsdiagramm beschreibt eine hypothetische Maschine, die sich zu jedem Zeitpunkt in einer Menge endlicher Zustände befindet.

Sie bestehen aus:

- einem Anfangspunkt
- einer endlichen Anzahl von Zuständen
- einer endlichen Anzahl von Ereignissen
- einer endlichen Anzahl von Transitionen, die den Übergang eines Objektes von einem in den anderen Zustand beschreiben
- einen oder mehrere Endpunkt

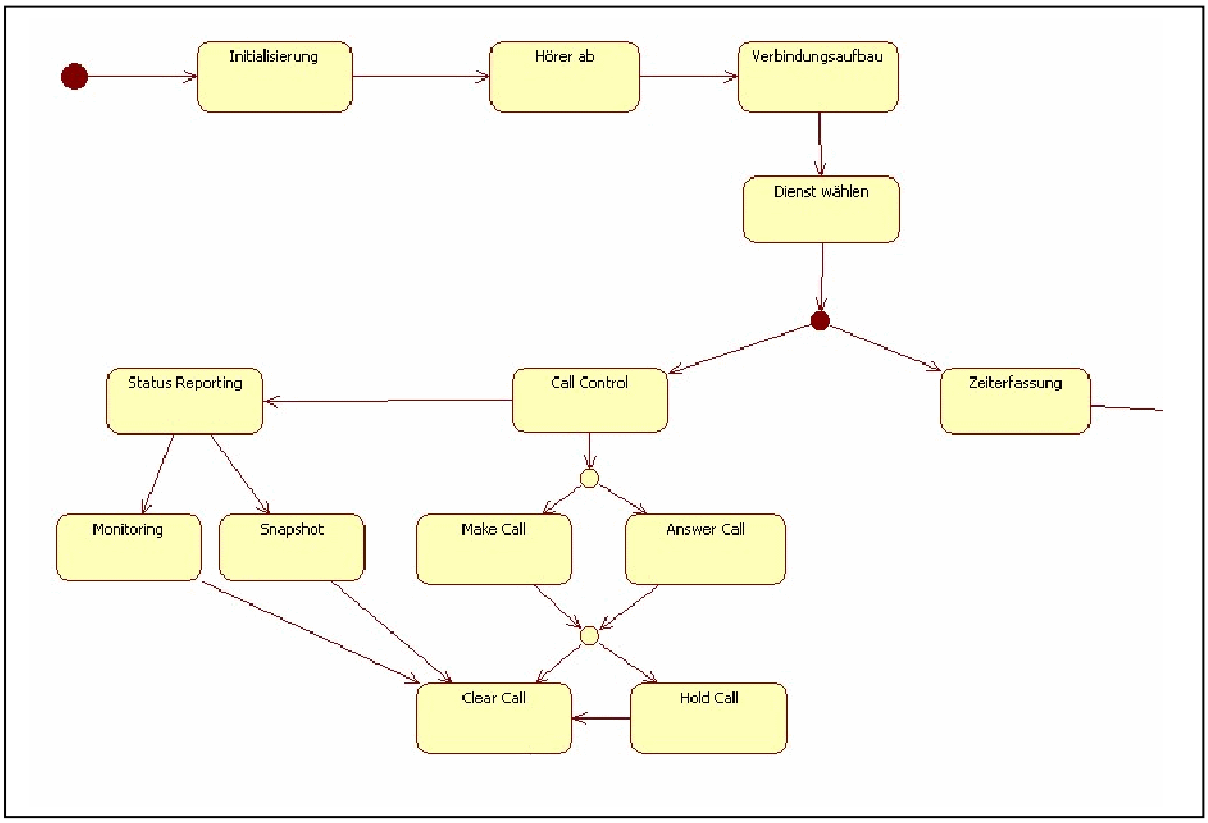


Abbildung 16: Beispiel für ein Zustandsdiagramm [4]

In der obigen Grafik ist zu erkennen, dass sich dem Ausgangspunkt, nur der Zustand Initialisierung anschließen kann. Dass auf einem Zustand auch zwei mögliche Zustände folgen können, ist am Zustand „Dienst wählen“ ersichtlich. Ihm kann der Zustand „Call Control“ oder „Zeiterfassung“ folgen. Nur einer von beiden kann erreicht werden. Dafür ist ein bestimmtes Ereignis, wie in diesem Beispiel das Drücken einer Ziffer, notwendig.

Dieser Diagrammtyp hat den Vorteil, dass komplizierte Prozessabläufe übersichtlich dargestellt werden können.

2.4.2.3 Aktivitätsdiagramm[11]

In einem Aktivitätsdiagramm werden die Objekte eines Programmes mittels der Aktivitäten, die sich während des Programmablaufes ereignen, beschrieben. Eine Aktivität ist ein einzelner Schritt innerhalb eines Programmablaufes, die einen speziellen Zustand eines Modellelementes, eine interne Aktion sowie eine oder mehrere von ihm ausgehende Transitionen enthalten. Gehen mehrere Transitionen von der Aktivität aus, so müssen diese mittels Bedingungen voneinander zu unterscheiden sein. Somit gilt ein Aktivitätsdiagramm als Sonderform eines Zustandsdiagrammes, dessen Zustände der Modellelemente in der Mehrzahl als Aktivitäten definiert sind.

Beschreibung

In einem Programmablauf durchläuft ein Modellelement eine Vielzahl von Aktivitäten, d.h. Zustände, die eine interne Aktion und mindestens eine daraus resultierende Transition enthalten. Die ausgehende Transition impliziert den Abschluss der Aktion und den Übergang des Modellelementes in einen neuen Zustand bzw. eine neue Aktivität. Diese Aktivitäten können in ein Zustandsdiagramm integriert werden oder aber in einem eigenen Aktivitätsdiagramm visualisiert werden. Ein Aktivitätsdiagramm ähnelt in gewisser Weise einem prozeduralem Flussdiagramm. Jedoch sind alle Aktivitäten eindeutig Objekten zugeordnet, d.h. sie sind entweder einer Klasse, einer Operation oder einem Anwendungsfall eindeutig untergeordnet.

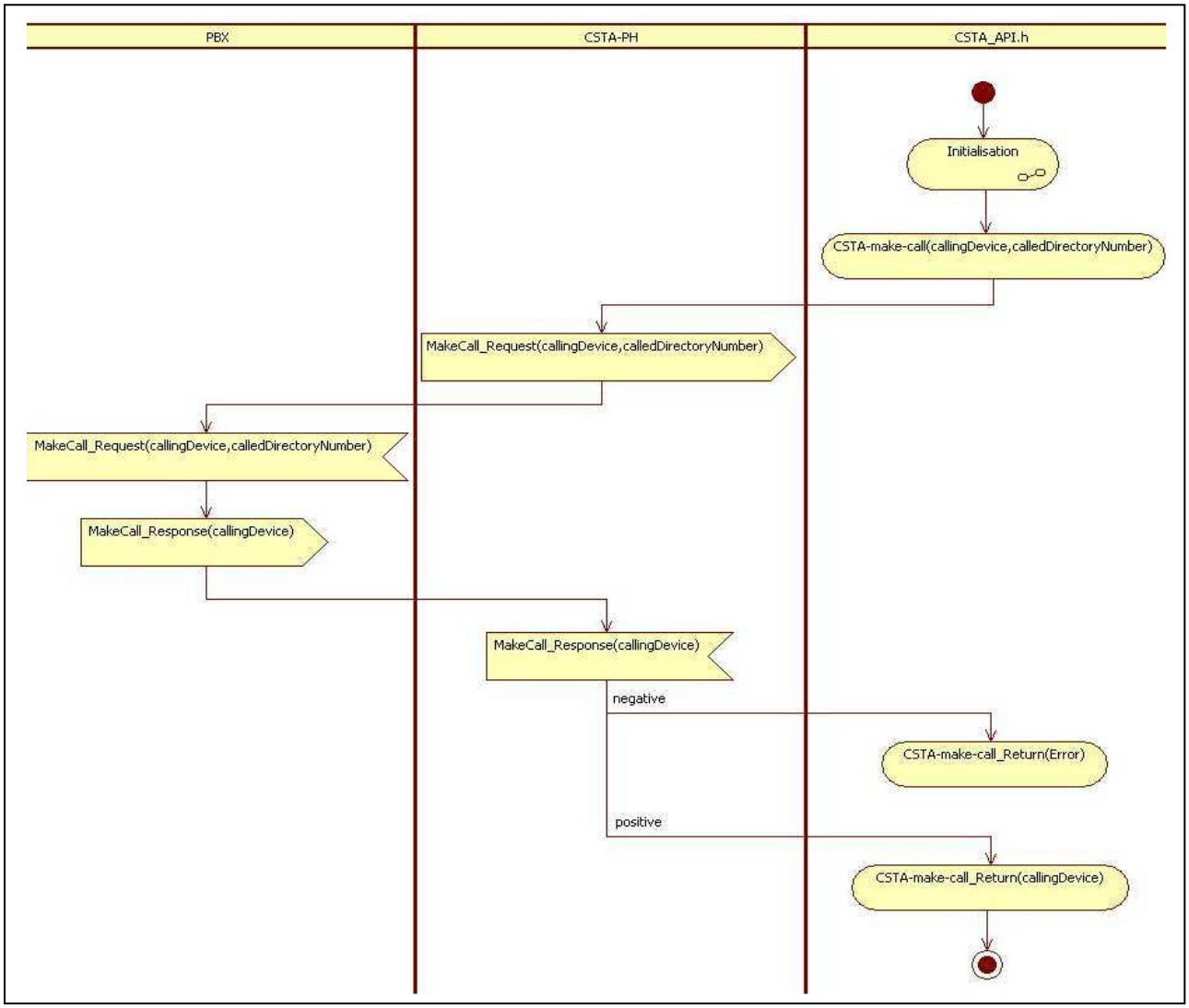


Abbildung 17: Beispiel für ein Aktivitätsdiagramm [4]

In der oberen Grafik ist zu erkennen, dass in der Header-File CSTA_API Methoden definiert sind, wie "Initialisation" oder "CSTA-make-call". Diese werden durch eine Applikation (CSTA-PH) aufgerufen und mit Werten gefüllt und an eine PBX geschickt. Es muss stets ein Anfangs- sowohl auch ein Endpunkt vorhanden sein.

Wie aus dem obigen Beispiel ersichtlich ist, kann durch diesen Diagrammtyp der Nachrichtenaustausch mehrere Systeme untereinander übersichtlich dargestellt werden.

3 Vergleich der ASN.1 Compiler

3.1 Software

Für die Betriebsweise des Treibers wird nur eine überschaubare Anzahl von Funktionen implementiert. Diese sind nachfolgend aufgelistet.

Call Control:

- Make Call
- Hold Call
- Clear Call
- Answer Call
- Park Call
- Join Call
- Alternate Call
- Conference Call

In der folgenden Tabelle wird auf die Bedeutung der einzelnen Funktionen eingegangen.

Funktion	Beschreibung
Make Call	Diese Funktion ermöglicht das Aufbauen einer Verbindung zwischen einem angerufenen und anrufenden Teilnehmer (Device).
Hold Call	Diese setzt eine vorhandene Verbindung auf Warten.
Clear Call	Clear Call trennt die Verbindung von allen Devices von einer bestimmten Verbindung.
Answer Call	Answer Call ist eine Funktion, um einen eingehenden Anruf entgegen zunehmen oder einen Anruf aus der Warteschleife anzunehmen, hauptsächlich dann von Interesse an Arbeitsplätzen mit nur Speaker-Units und Headsets für Hand-Free Operations.
Park Call	Park Call verbindet in ein bereits vorhandenes Gespräch eines Devices mit einer geparkten Device und erzeugt somit eine neue Verbindung.
Join Call	Join Call ermöglicht an einem vorhandenen Gespräch ein weiteres Device teilnehmen zu lassen.
Alternate Call	Diese Funktion setzt eine aktive Verbindung auf Hold, um sich mit einem anderen Device zu verbinden, das gerade wartet oder anruft.
Conference Call	Conference Call verbindet mehr als 2 Devices zu einem Call.

Tabelle 8: Bedeutung der Call Control Funktionen

Monitoring Service:

- Monitor Start
- Monitor Stop

Funktion	Beschreibung
Monitor Start	Monitor Start startet an der PBX einen Event Report für einen Call, einem Device oder einem Device, das an mehreren Calls beteiligt ist.
Monitor Stop	Monitor Stop stoppt diesen Event Reporter Dienst.

Tabelle 9: Bedeutung Monitor Service Funktionen

Snapshot Service:

- Snapshot Call
- Snapshot Device

Funktion	Beschreibung
Snapshot Call	Diese Funktion liefert Informationen über die Devices in dem ausgewählten Call, außerdem gibt sie Aufschluss über Informationen des Device Identifiers, ihre Verbindung in dem Call und die lokalen Verbindungszustände.
Snapshot Device	Snapshot Device liefert Informationen über die Verbindungen eines ausgewählten Devices zurück, außerdem gibt sie Aufschluss über die ausgegebenen Informationen des Call Identifiers und darüber, in welchem Zustand sich das Device in diesem Call befindet.

Tabelle 10: Bedeutung Snapshot Service Funktionen

I/O Service:

- IO Register
- Start Data Path
- Stop Data Path
- Data Path Resumed
- Data Path Suspended
- Send Data
- Multicast Data
- Send Broadcast Data

Funktion	Beschreibung
IO Register	IO Register registriert eine Applikation als IO-Server für eine ausgewähltes Device oder für alle Device in einer Subdomäne.
Start Data Path	Start Data Path stellt einen Datenzugang her und setzt sich in den Zustand auf Open.
Stop Data Path	Stop Data Path beendet einen Datenzugang und setzt sich in den Zustand „NULL“.
Data Path Resume	Data Path Resume setzt den Datapath in den Zustand Suspended und kann diese durch die Funktion DataPathResumed wieder in den Zustand Open gebracht werden.
Data Path Suspend	Data Path Suspend setzt eine Datenverbindung vom Zustand Open in Suspended.
Send Data	Bei Send Data ist die Voraussetzung das eine DataPath vorhanden ist, sendet und empfängt Daten von einem CSTA Objekt.
Multicast Data	Multicast Data sendet Daten an mehrere speziell ausgewählte Datenpaths.
Send Broadcast Data	Send Broadcast Data sendet Daten an alle offenen Datenpaths einer Applikation.

Tabelle 11: Bedeutung I/O Service Funktionen

System Services:

- System Status
- System Register

Funktion	Beschreibung
System Status	System Status liefert Informationen über die PBX und deren derzeitigen Status.
System Register	Es muss erst eine System Request gesendet werden, erst danach wird ein System Status Request von der PBX verarbeitet.

Tabelle 12:Bedeutung System Service Funktionen

3.2 Entwicklungsumgebung

Eine Zielsetzung war es, einen Treiber zu entwickeln, dessen Programmcode plattformunabhängig einsetzbar ist. Es müssen bestimmte Regeln beim Erstellen von Portablen Code beachtet werden.

Dabei soll plattformunabhängig als die Eigenschaft eines Programmes aufgefasst werden, mit verschiedener Hardware und Software Konstellationen kompatibel zu sein. Dabei ist es wichtig, dass der Code auf unterschiedliche Betriebssysteme, wie Mac OS, Windows Betriebssysteme oder Linux Betriebssysteme und mit verschiedenen Rechnerarchitekturen, wie Macintosh oder Windows, lauffähig ist.

Am Grad der Portabilität des Quellcodes kann man die Plattformunabhängigkeit messen. Dabei hängt der Grad der Portabilität vom notwendigen Aufwand ab, der benötigt wird, um den Quellcode an die neue Plattform anzupassen. Ist kein Aufwand notwendig, dann ist die Portabilität zu 100 Prozent gegeben.

Da in dieser Diplomarbeit der entstehende Quellcode sowohl unter einem Linux System als auch auf einem Windows System kompilierbar sein soll, muss bei dem Programmieren darauf geachtet werden, dass nur Bibliotheken eingebunden werden, die von beiden Compilern verwendet werden können.

Einige Möglichkeiten die Portabilität des Quellcodes zu erhöhen, sind[2]:

- Wo immer möglich, symbolische Konstante (mittels #define) benutzen
- Alles explizit deklarieren. Funktionen, die keinen Funktionswert liefern, sind mit `void` zu deklarieren.
- Die Funktion main immer mit einem expliziten return oder exit verlassen.
- Nie annehmen, int und Pointer hätten dieselbe Größe.
- Alle maschinenabhängigen Deklarationen und Definitionen in einem Header – File konzentrieren
- Code vermeiden, der auf Annahmen über irgendeine Ordnung basiert.
- Nie int verwenden. Eine int hat die gleiche Größe wie ein Register der Maschine. Wo immer diese Größe eine Rolle spielt, ist short oder long zu verwenden. Am besten mit einem Header – File und typedef arbeiten.
- NULL sollte nicht als explizites Argument für irgendeine Funktion, welcher Pointer entgegennimmt, verwendet werden. In gewissen Umgebungen sind Pointer auf verschiedenen Datentypen unterschiedlich groß.

Es ist kaum möglich, einen hundertprozentigen portablen Code zu erstellen. Deshalb sollte man die maschinen- und betriebssystemspezifischen Programmteile in separate Files zusammenfassen.

Die Entwicklung des Treibers erfolgte auf einem Windows XP Professional Service Pack 3 PC.
Als Entwicklungsumgebung wurde Visual Studio 2005 Version 8.0.5 gewählt.

Bei der Recherche nach möglichen ASN.1 Compilier wurden folgende Produkte von Object Systems, Marben, OSS, Lev Walkin und III ASN.1 Tool nach den aufgeführten Kriterien getestet.

- Aktualität
- Kosten
- Unterstützte Betriebssysteme
- Bedienung
- mögliche nutzbare Encoding Rules
- Umsetzen des CSTA Standards in ASN.1 Notation
- Unterstützte Programmiersprachen

OSS ASN.1 Tool [20]

Als Erstes soll das Tool von OSS betrachtet und bewertet werden.

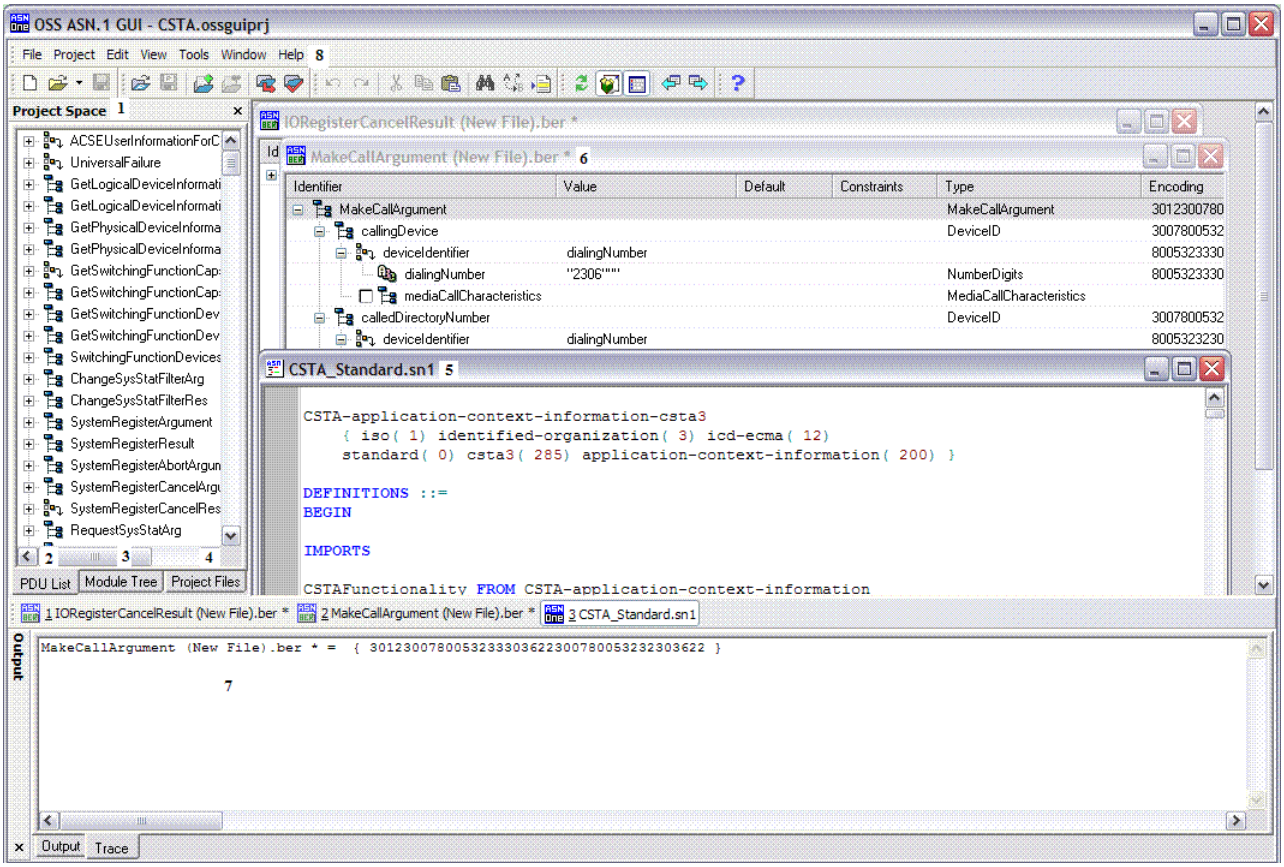


Abbildung 18: OSS ASN.1 GUI

In der oberen Grafik ist ein Ausschnitt der OSS ASN.1 GUI dargestellt.
Dabei beschreibt 1 das Project Space, dies dient für die Anzeige von PDU List, Modul Tree und Project Files.
Mit der PDU–List (2) werden alle erzeugten PDU's aufgelistet, die durch das Compilieren des ASN.1 Standards entstanden sind.
Mittels des Module Tree (3) können die erzeugten Module angezeigt und ausgewählt werden.
Im Bereich Project Files (4) werden die verwendeten ASN.1 Files aufgelistet.
Das Fenster Tutorial.asn (5) dient zur Anzeige des ASN.1 Quellcodes.
Mit dem Fenster, markiert mit der Zahl 6, kann die Struktur einer ausgewählten PDU dargestellt und gefüllt werden.
Das Fenster (7) dient zur Ausgabe von Projektinformationen, beispielsweise werden darüber die Fehler oder Warnungen die beim Compilieren des Standards aufgetreten sind, dargestellt.

[20] www.oss.com

Die 8 markiert die Menüzeile, in der alle relevanten Einstellungen vorgenommen werden, sowie auch laden oder speichern aller für das Projekt wichtigen Dateien.

Das Tool verfügt über einen ASN.1 Compilier und einer Laufzeitbibliothek.
Der Einsatz des Compilier wird anhand folgender Abbildungen erklärt.

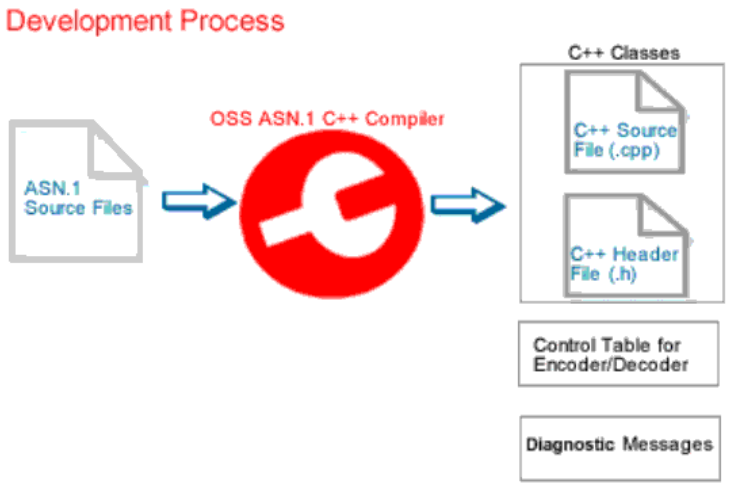


Abbildung 19: Development Process Quelle: [12]

In der oberen Abbildung ist der Entwicklungsprozess dargestellt. Als Ausgangspunkt sind nur die ASN.1 Source Files vorhanden. Aus ihnen entstehen mittels OSS Compiler Klassen in der gewünschten Programmiersprache. Für die Umsetzung der Diplomarbeit sind C++ Klassen notwendig. Für die Klassen wird eine .cpp mit dazugehörigen Header Files angelegt. In den .cpp Files werden Methoden definiert, die es ermöglichen, den Strukturen Werte zuzuweisen und zu encodieren. Außerdem wird eine Control Table für Encoding und Decoding erzeugt.

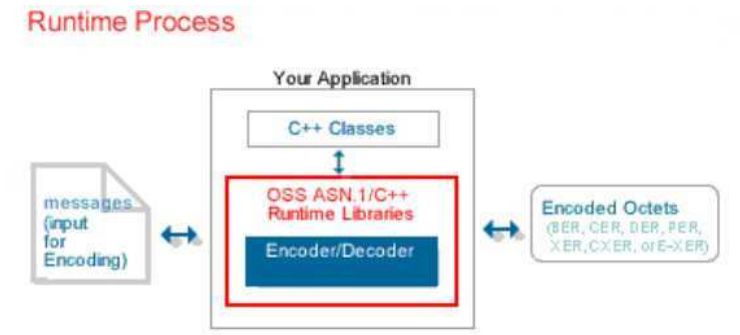


Abbildung 20: Runtime Process Quelle:[12]

Anhand der Abbildung 23 wird der Runtime Prozess erklärt. Die Applikation nutzt die generierten Klassen und die Laufzeit Bibliothek. Wird eine Message an die Applikation übergeben, wird diese gefüllt und durch die Encoding Methoden nach bestimmten Richtlinien (BER, XER, CER) in einen Oktettstrom umgewandelt. Die Decodierung ist auch möglich. Dabei wird ein eingegangener Stream durch Decoding Methoden wieder in eine gefüllte Struktur umgewandelt, verarbeitet und / oder als Message ausgegeben.

Marben ASNSDK TCE Tool[21]

Nachfolgend soll das Tool von Marben betrachtet werden.

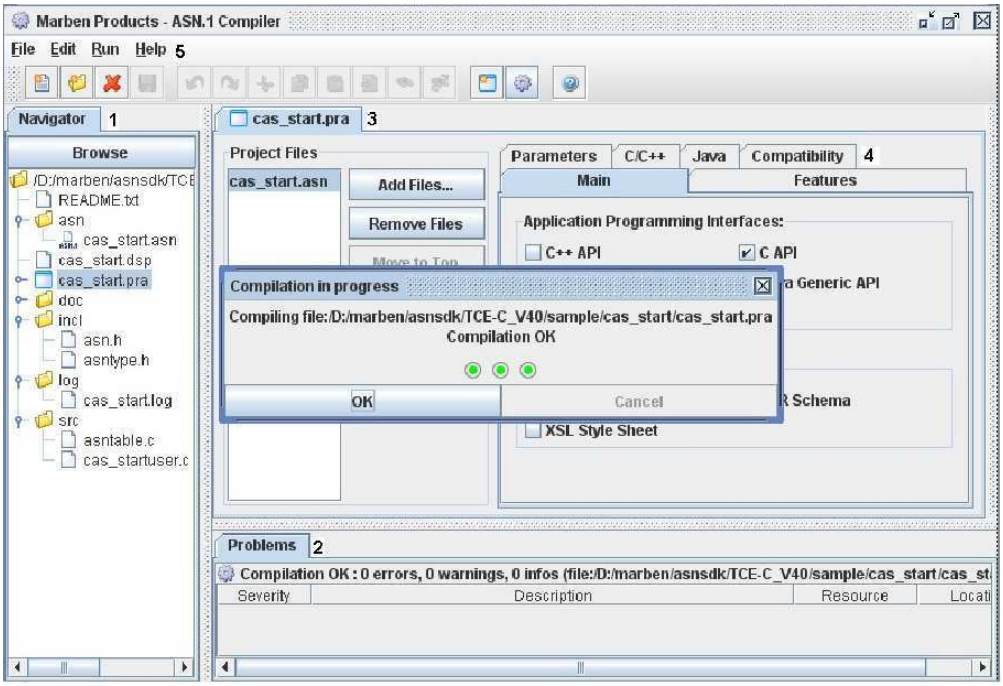


Abbildung 21: Marben ASNSDK TCE TOOL - Programmausschnitt

In der Oberen Abbildung ist ein beispielhafter Programmausschnitt des Marben Tools dargestellt.
Bei der Markierung 1 befindet sich der Navigator. Darin sind alle Projektdateien aufgelistet.
Im Fenster Problems (2) werden alle Warnungen und Fehler, die bei der Compilierung entstehen, ausgegeben.
Im Fenster Project Files (3) können die ASN.1 Files dem Projekt hinzugefügt werden.
Bei der Markierung 4 werden alle projekt- und compilerabhängigen Einstellungen vorgenommen, z.B. in welcher Programmiersprache die Klassen und Methoden entstehen sollen.
Bei 5. befindet sich die Menüleiste, um Einstellungen zu speichern oder Projekte zu laden.

Dem Produkt wird ebenfalls eine Laufzeitbibliothek mitgeliefert. Der Einsatz des Compilers und die Verwendung in einer Applikation erfolgt nach dem gleichen Prinzip wie bei OSS.

[21] www.marben-products.com

Objective Systems ASN1C Compiler

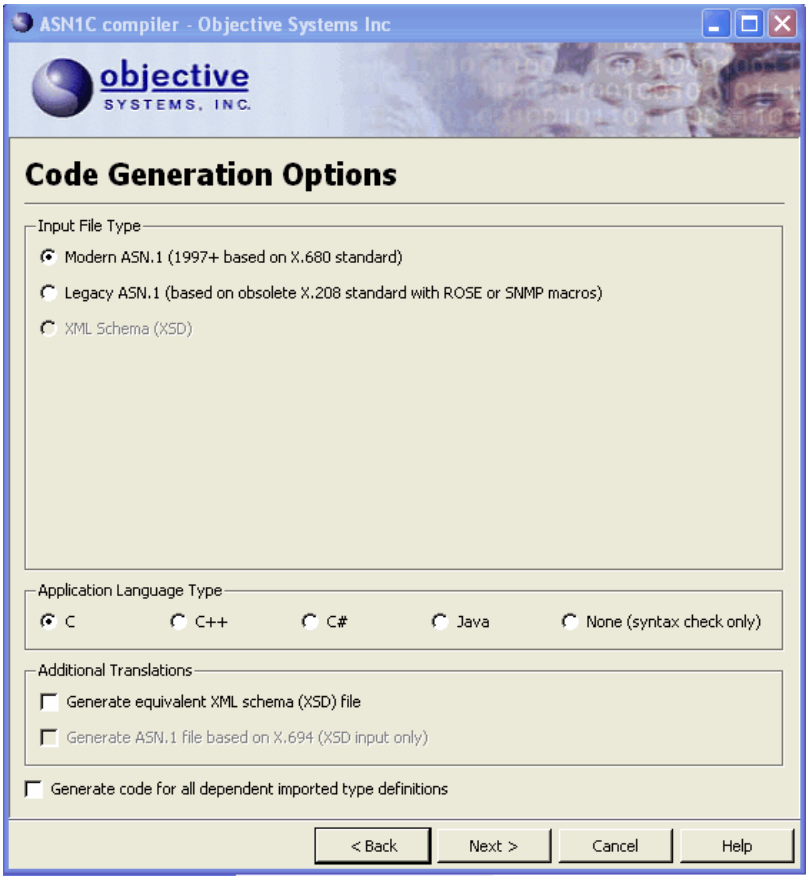


Abbildung 22: Object Systems ASN1C Compiler- Programmausschnitt

In der obigen Abbildung ist ein Ausschnitt aus der GUI vom ASN1C Compilier dargestellt. Das Programm kann mittels GUI oder per Kommandozeile verwendet werden.
Die Dokumentation ist auf der Herstellerseite sehr ausführlich und ermöglicht somit einen schnellen Einstieg in die Benutzung. Die grafische Übersicht des Programms ist sehr schlicht gehalten. Das Programm startet mit einem Fenster und durch das Klicken auf Next durchläuft man eine Reihe von Einstellungsmöglichkeiten, am Ende werden die Klassen und die notwendigen Methoden erstellt.
Der ASN1C Compiler wird mit einer Laufzeitbibliothek verkauft.

Zur Veranschaulichung wird folgend eine Schritt-für-Schritt Anleitung für diese Software gegeben.

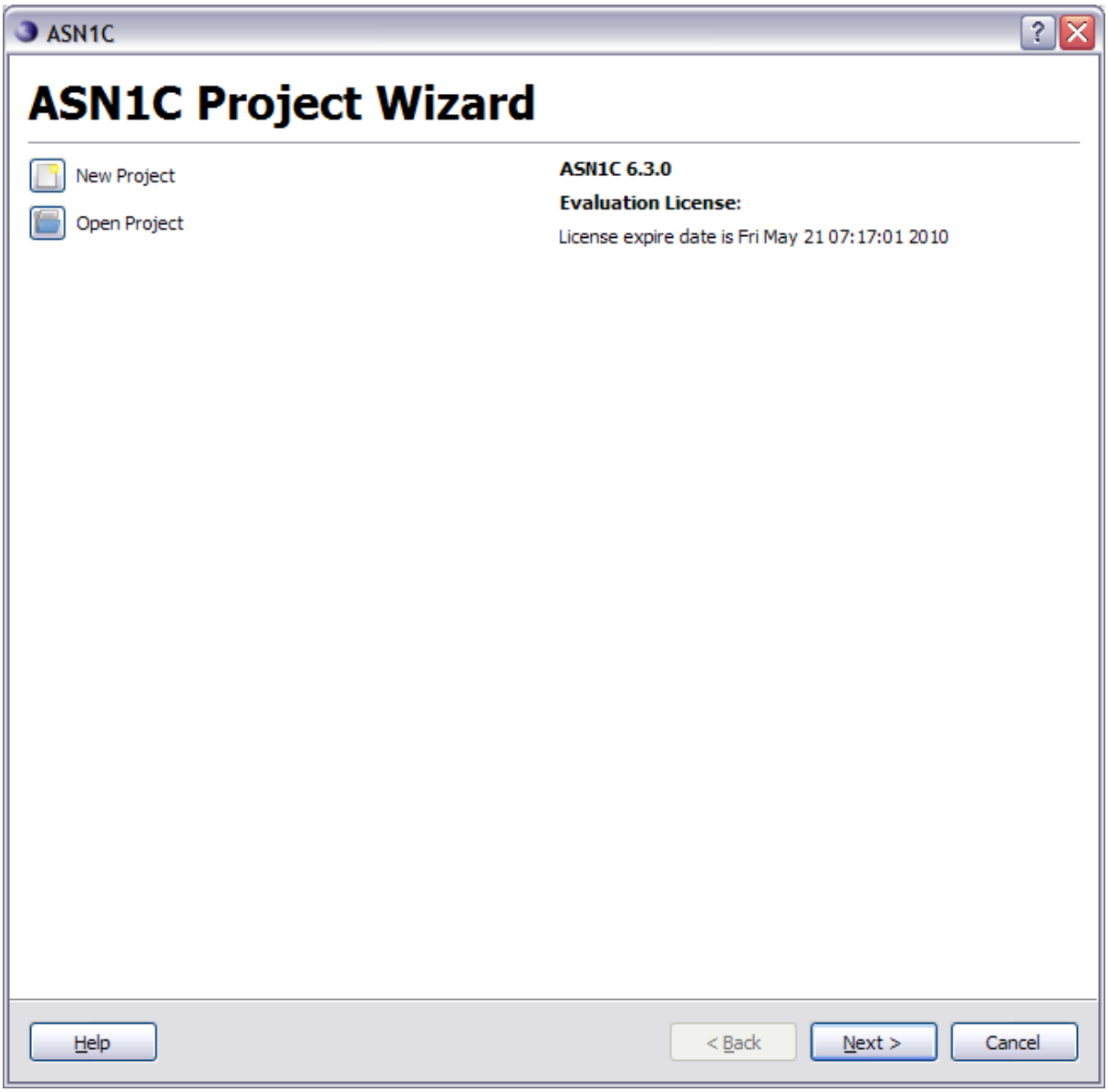


Abbildung 23: ASN1C Startbildschirm

In der Abbildung 23 ist der Startbildschirm zu sehen. Hier kann man wählen ob man ein bereits vorhandenes Projekt öffnen will oder ein neues Projekt anlegen möchte.

Für diese Anleitung wird ein neues Projekt angelegt. Es soll der CSTA-Standard der auf der Internetseite der Standardisierungsbehörde ecma-international heruntergeladen werden.

Durch Klicken auf New Projekt gelangt man auf die neue Oberfläche,

zu sehen in Abbildung 24.

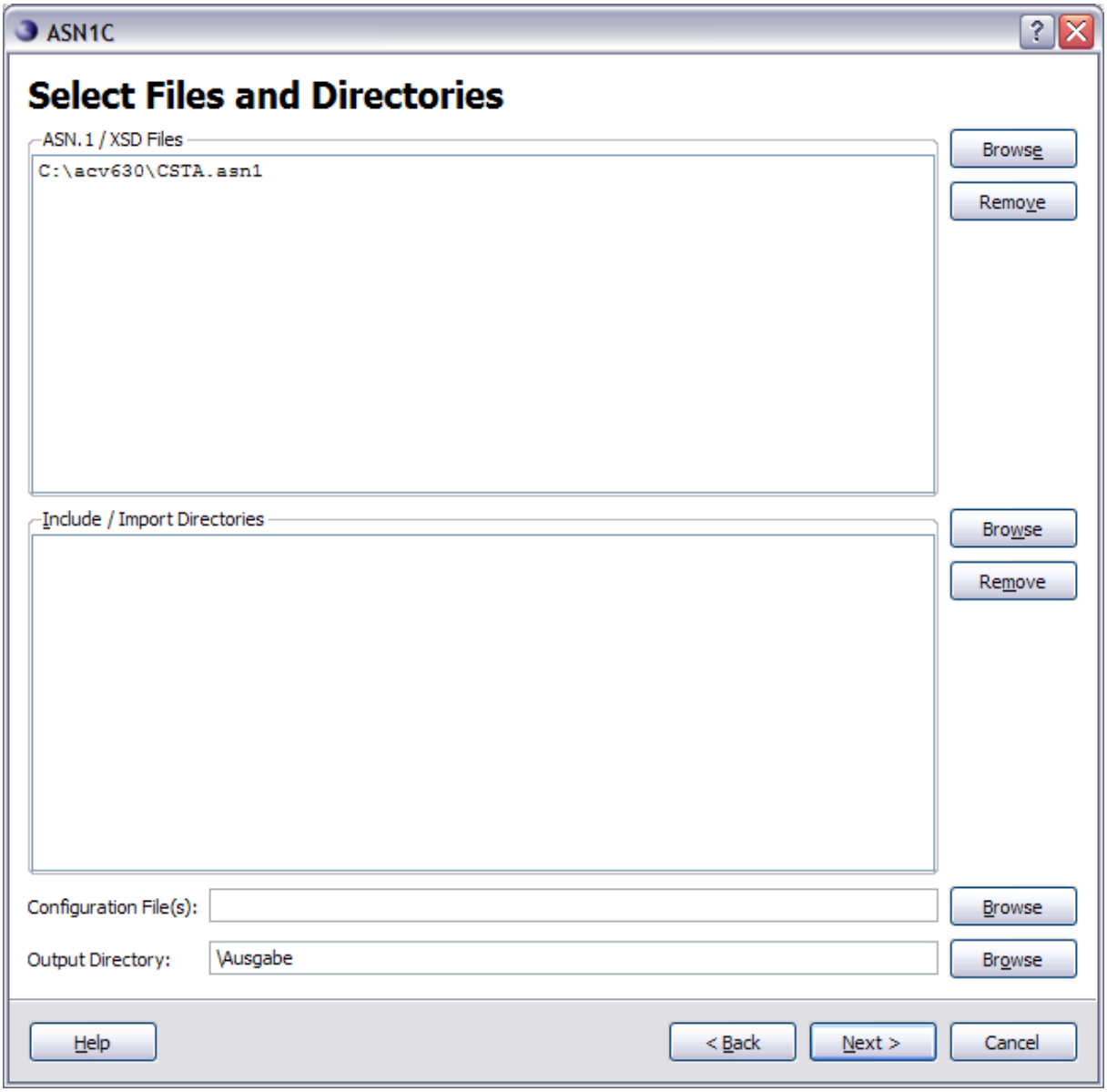


Abbildung 24:ASN1C ASN.1 Dateien hinzufügen

In diesem Abschnitt werden die ASN.1 Dateien dem Projekt hinzugefügt, außerdem lassen sich weitere Ordner und Configuration Files dem Projekt beigefügt werden.

Eine wichtige Einstellungsmöglichkeit ist die Wahl des output Directory.

Durch Klicken auf Next kommt man zu den Nächsten Einstellungen.

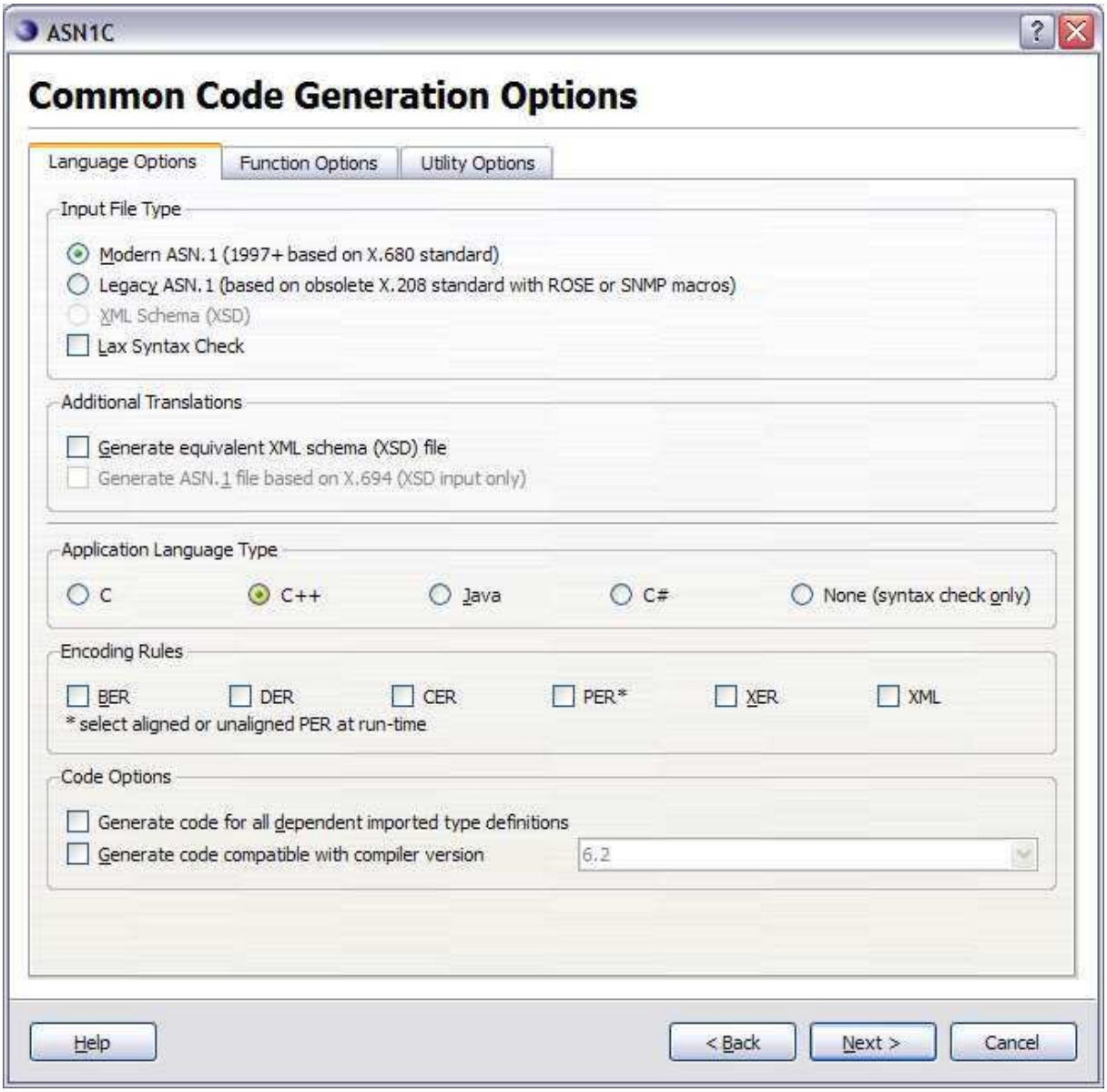


Abbildung 25: ASN1C Language Option

In der Language Option lässt sich eine Vielzahl an Einstellungen vornehmen, z.B. Auswahl der verwendeten Encoding Rules, die Art der Input Datentypen, oder die zu nutzende Programmiersprache.

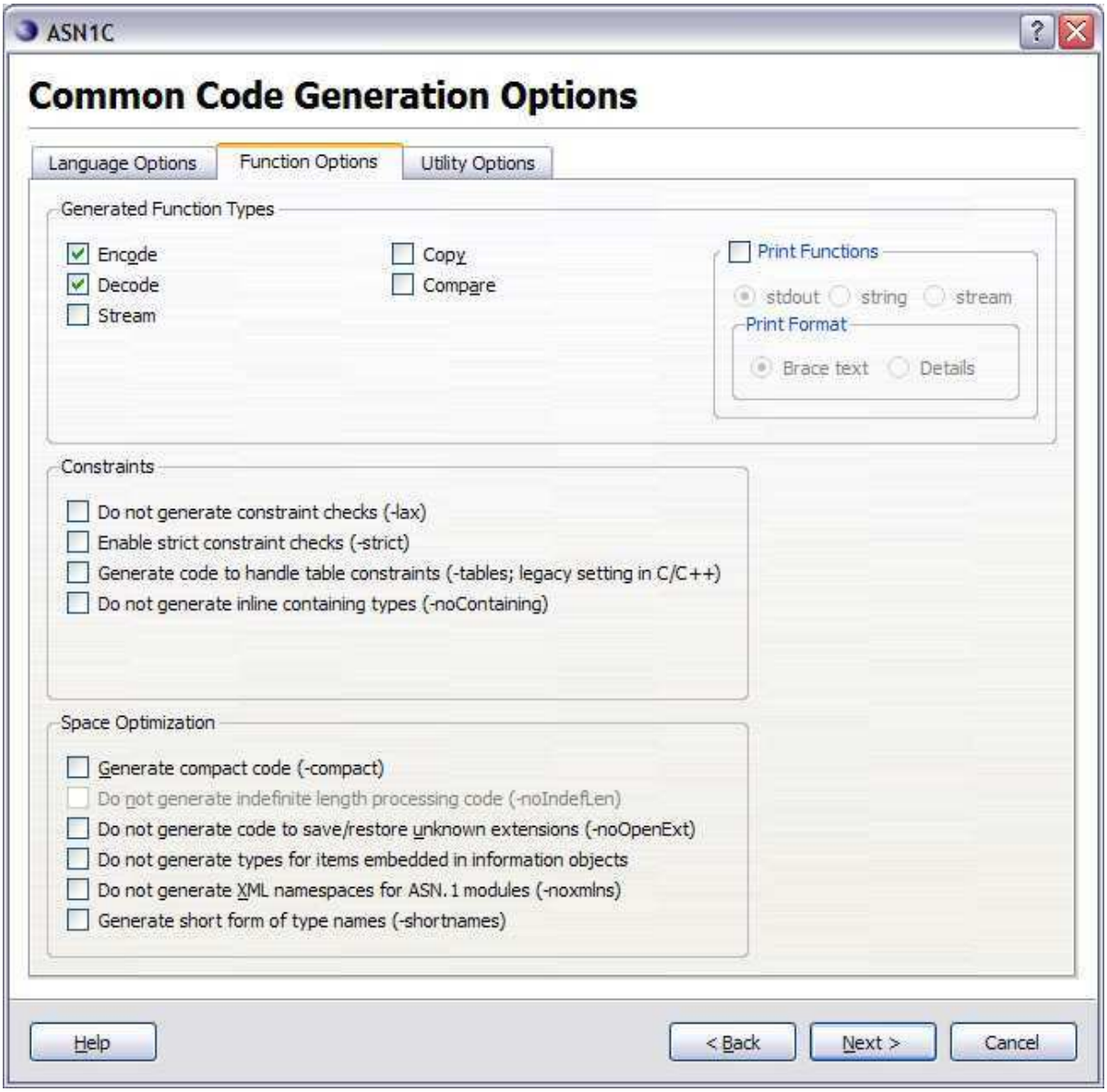


Abbildung 26:ASN1C Function Options

Mit den Function Options , siehe Abbildung 26, lassen sich Einstellungen vornehmen über die erwünschten Hilfsfunktionen. Es können Encode, Decode, Stream , Copy und Compare Funktionen durch den ASN.1 Compiler miterstellt werden.

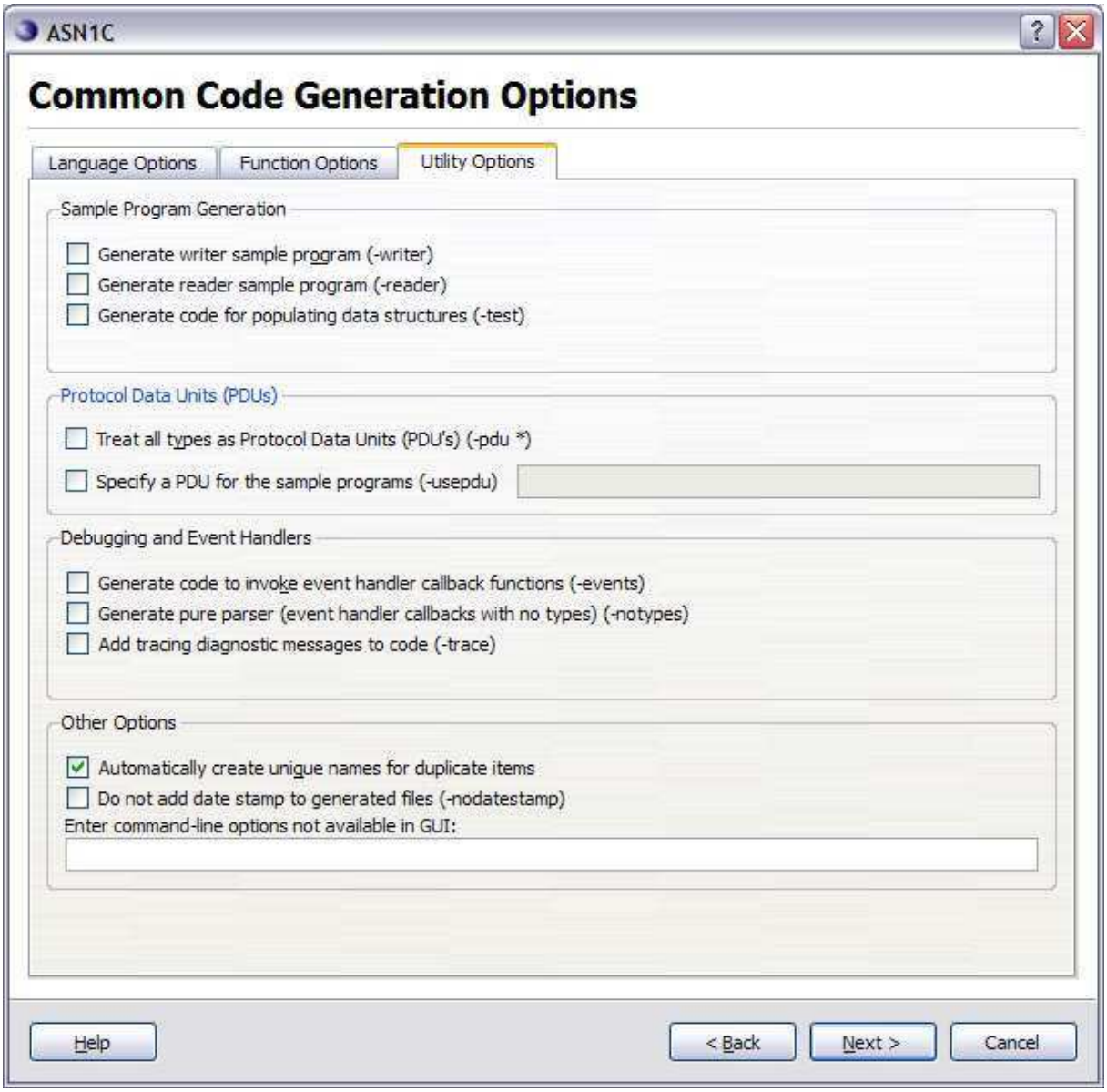


Abbildung 27: ASN1C Utility Options

Mittels den Utility Options hat der Nutzer die Möglichkeit sich writer,reader und test programme erstellen zu lassen. Diese Programme dienen dazu den zu erstellenden Code besser nutzen zu können, da in den sample Programms eine Möglichkeit der Verwendung des Codes gegeben wird.

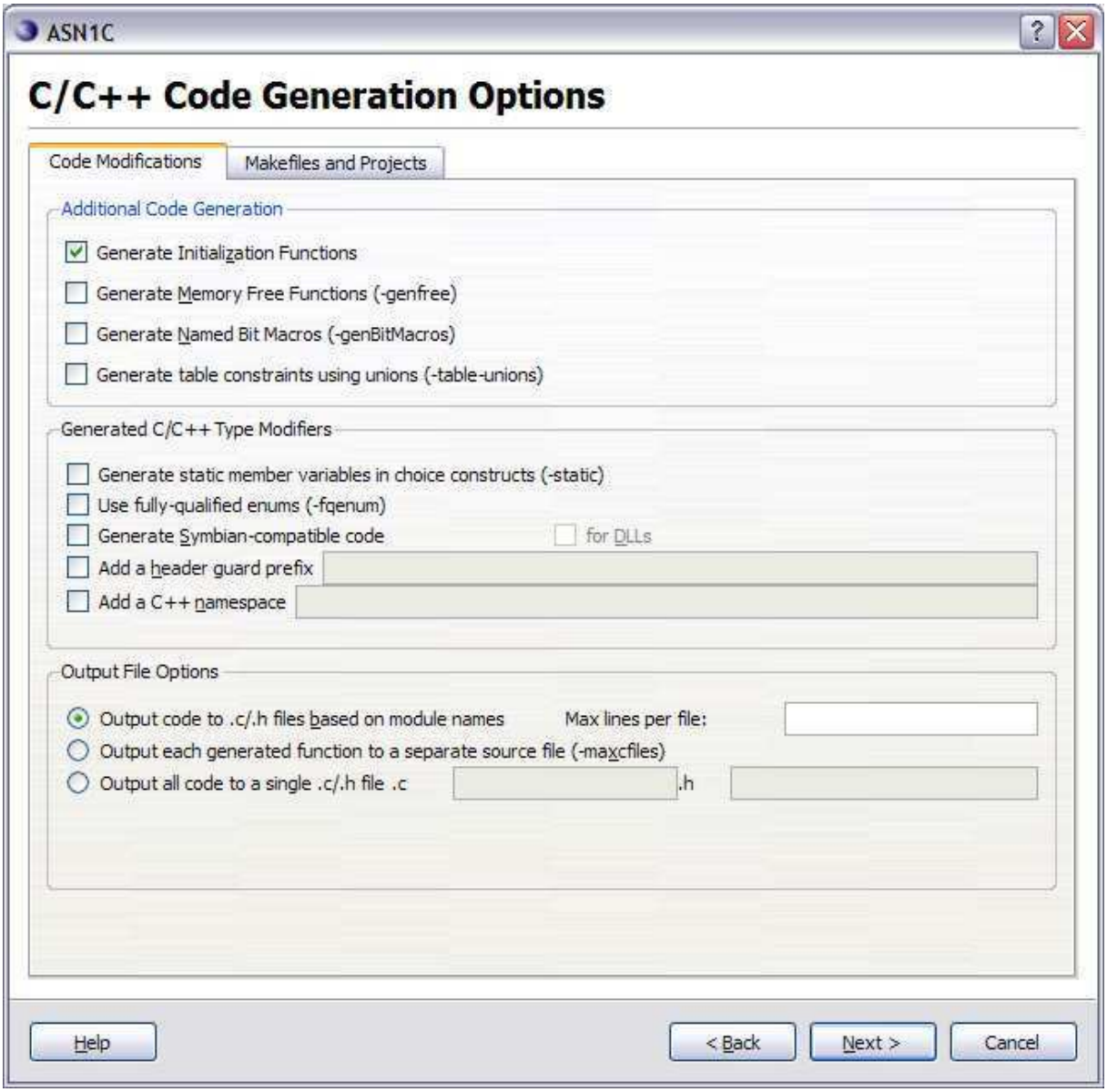


Abbildung 28: ASN1C Code Modification

Mittels den Möglichkeiten im Fenster Code Modifications lässt sich der zu erstellende Code auf die Individuellen Wünsche anpassen. Für genauere Bedeutung der einzelnen Punkte sollte die Dokumentation des Herstellers herangezogen werden.

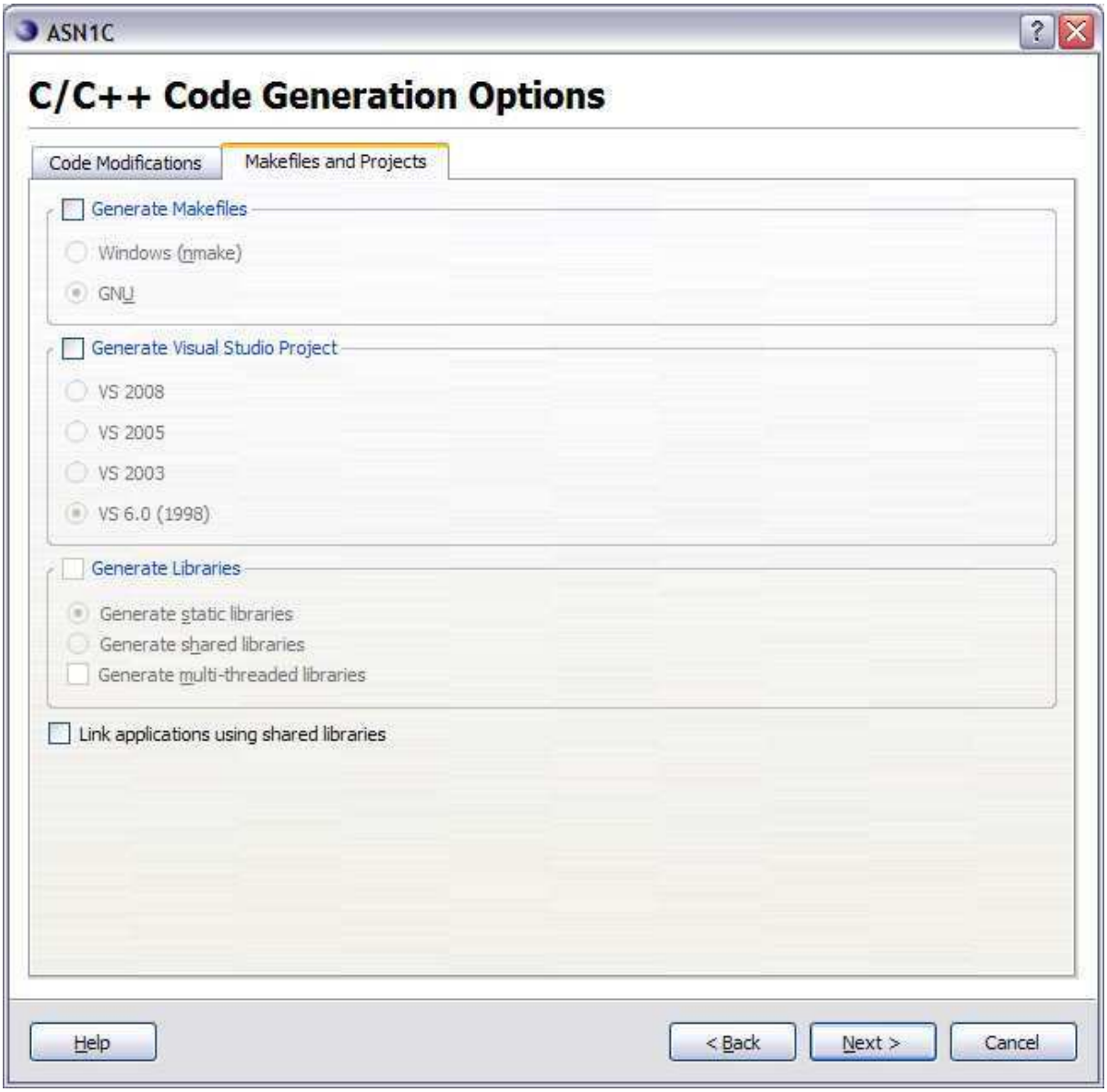


Abbildung 29: ASN1C Makefiles and Projects

Im Fenster Makefiles and Projects kann zum Abschluss ausgewählt warden für welchen Compiler ein Makefile erstellt warden soll und für den Fall es soll für das spätere Projekt ein Visual Studio Project verwendet werden kann gleich für die Version passenden Projekt angelegt werden.

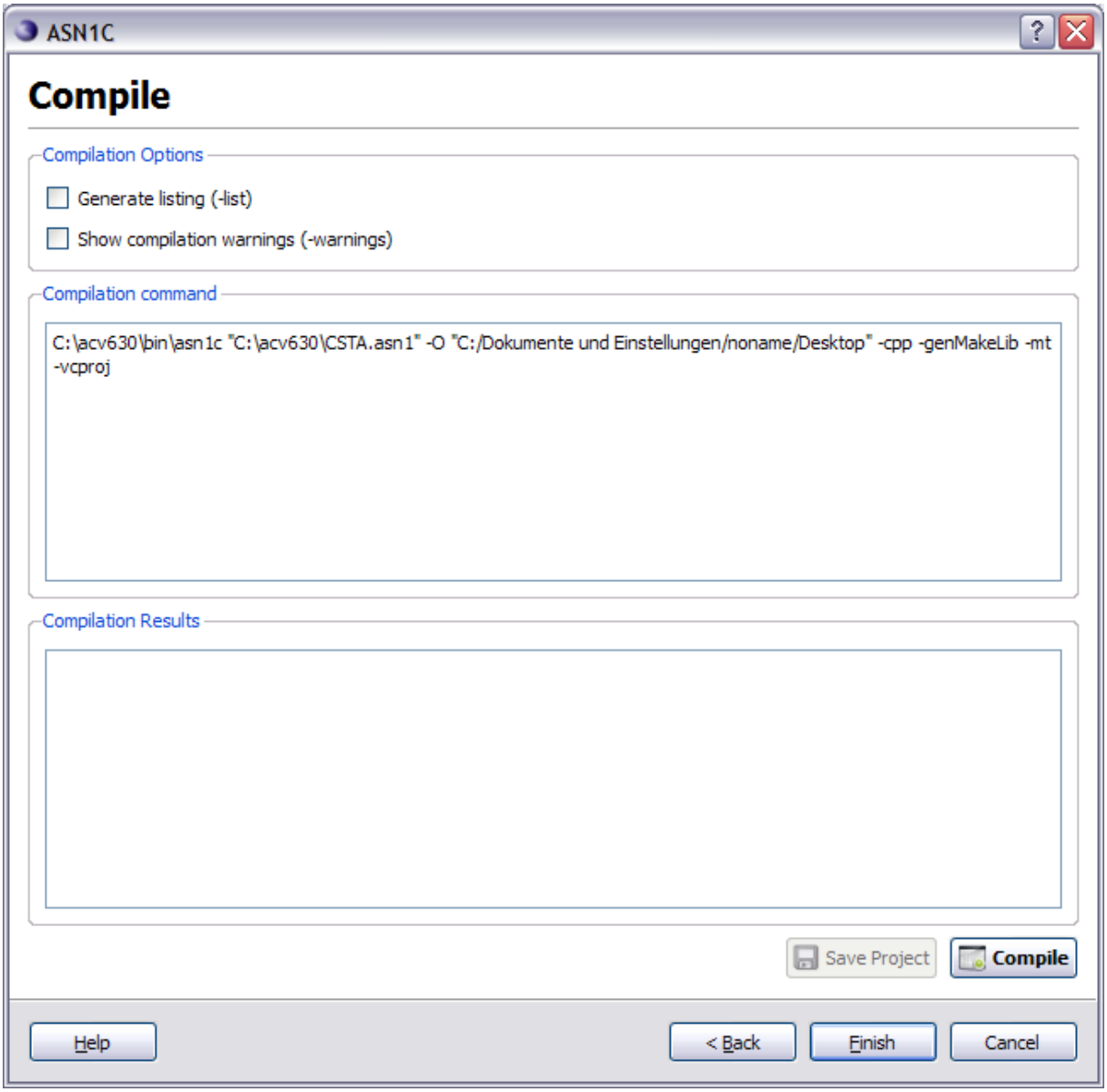


Abbildung 30: ASN1C Compile

Die obere Abbildung gibt eine Zusammenfassung auf alle Compiler Einstellungen. Über das Compilation Results Fenster erfolgt die Ausgabe von Information über mögliche Fehler während des Compilierens.

Das Generieren mit dem Tool ASN.1C Compiler von Object Systems generierte 929 C++ Files und Header(233 - Files). Aber nur eine geringe Auswahl der erzeugten Files wird später für die Umsetzung des Projektes benötigt werden.

Lev Walkin ASN.1 Tool

Lev Walkin ASN.1 Tool ist eine Open Source Software Anwendung. Die Benutzung ist ausschließlich über die Kommandozeile möglich. Es steht keine Laufzeit Bibliothek zur Verfügung. Auch durch vielen Versuchen konnte kein nutzbarer Quellcode erzeugt werden.

III ASN.1 Tool

Diese Software ist eine Open Source Anwendung und wird über die Kommandozeile bedient. Es steht keine Laufzeit Bibliothek zur Verfügung.
Auch nach langer Einarbeitung war es dem Autor nicht möglich, einen verwendbaren Quellcode zu erzeugen. Aus diesem Grund kann dieser Compiler für das Projekt nicht verwendet werden.

Zusammenfassender Vergleich der Kompilier

	OSS ASN.1 Tool	Marben ASNSDK TCE	Lev Walkin ASN.1 Tool	III ASN.1 Tool	Objective Systems ASN1C
Kosten	5600$ eine License	Hochschul lizenze möglich	Open Source	Open Source	Hochschul lizenze möglich
Betriebssystem	Windows, Solar-SPARC, HP-UX, Linux	Windows, Linux	Linux, MacOS, Solaris 9, BSD, Windows	Linux, Windows	Windows, Linux
Ausgabeformat	C ,C++, Java, C#	C, C++, Java	C	C++	XML, C, C++, C#, Java
Encoding Rules	BER, CER, DER, PER, XER, CXER, EXER	BER, DER, PER, XER, CXER	BER, PER, XER	BER, PER, AVN	BER, DER, CER, PER
Code generiert	ja	nein	nein	nein	ja
Quelle	www.oss.com	www.marben-products.com	lionet.info/asn1c	iii-asn1.sourceforge.net	www.obj-sys.com

Fazit:

Für die Umsetzung des Diplomthemas wird eine überschaubare Anzahl an PDU's benötigt. Daher sollte analysiert werden, ob es notwendig ist, kommerzielle Software zu erstehen, oder ob auch Open Source Anwendungen alle notwendigen Bedingungen erfüllen.
Dafür muss abgeschätzt werden, wie hoch der Arbeitsaufwand wäre, wenn alle notwendigen Klassen selbst zu erstellen sind, um die Verwendung eines ASN.1 Compilier zu umgehen.

Da die Kosten für das Projekt einen der Hauptfaktoren darstellt, musste bei Programmen mit kostenpflichtigen Lizenzen auf deren Nutzung verzichtet werden, da diese den vorhandenen Kostenrahmen bei weitem überschreiten würden.
Die Entwicklung der Open Source Anwendungen wurde eingestellt, daher war es dem Autor nicht möglich, vorhandene Probleme mit den Entwicklern zu klären.
Aus diesem Grund und da die Anzahl und Komplexität der PDUs überschaubar ist, wurde sich für die Alternative der Selbstprogrammierung entschieden. Dies hat zwar einen höheren Arbeitsaufwand zur Folge, aber dafür würden keine laufenden Kosten anfallen.

3.2.1 Hardware

Aufbau der Implementierungsumgebung

Die DLL beinhaltet alle Funktionen bzw. Methoden, die für die Kommunikation zwischen PBX und der Applikation notwendig sind, um die oben erwähnten Dienste umzusetzen. Mittels eines kleinen Testprogrammes werden einzelne PDU´s aus der DLL aufgerufen und per Socket Anbindung an die HiPath4000 gesendet und die empfangenen Requests ausgewertet. Dadurch können schon zu Beginn mögliche Fehler in der Programmierung erkannt und behoben werden.

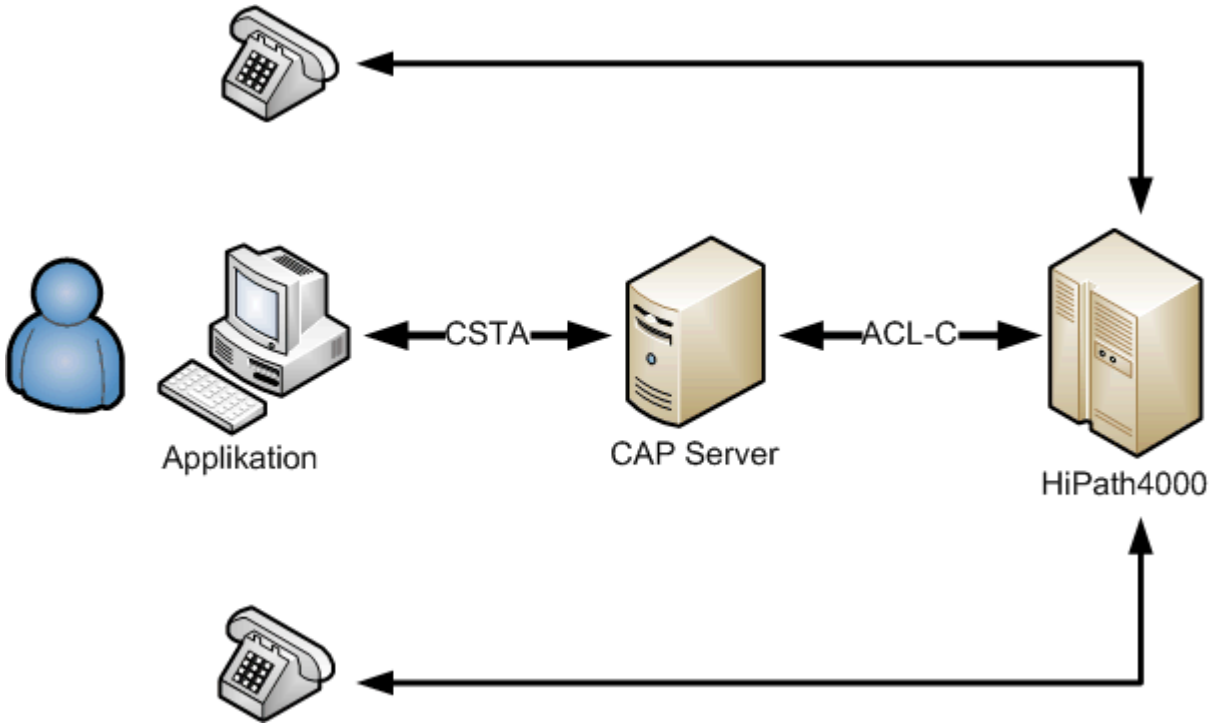


Abbildung 31: Hardware Umgebung

4 Modellierung und Konzeption des Treibers

4.1 Treiber

Die Treiberfunktionalität unterteilt sich allgemein in die Bereiche:

- Verbindungsaufbau und Verbindungsabbau mit der HiPath4000
- Encodierung und Decodierung der Nachrichten
- Anlegen und Befüllen der Strukturen

Die CSTA DLL wurde nach WOSA[22] konzipiert, dadurch wird es möglich das die Applikationen die HiPath4000 steuern können ohne Wissen auf das verwendete Protokoll zu habe, nur das Interface und die Unterstützen Funktionen sind wichtig,

Die CSTA.DLL enthält sowohl das Interface von Applikation zu HiPath4000 als auch das Interface HiPath zu Applikation.

[22] Windows Open System Architecture

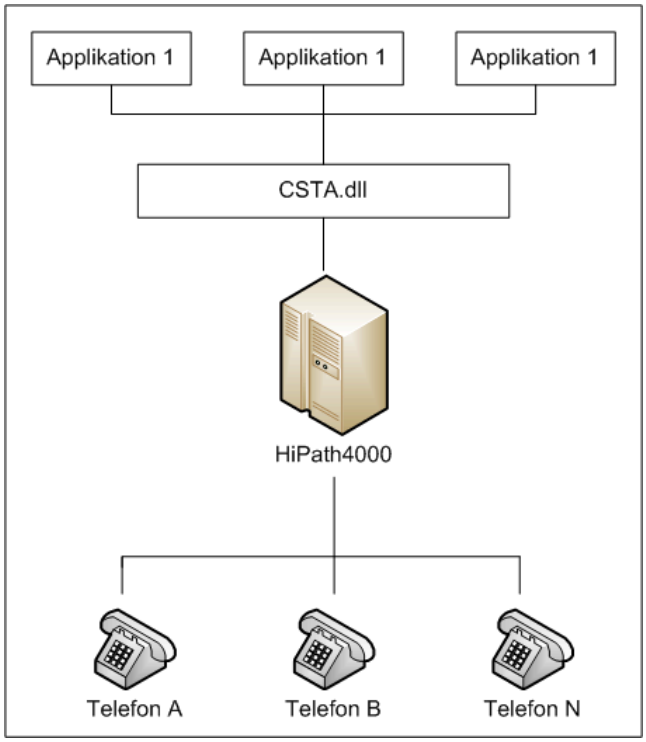


Abbildung 32: CSTA.dll

Mit Abbildung 32 sollen die Bestandteile des Treibers und deren Beziehung untereinander grafisch aufgezeigt werden. Über die Programmierschnittstelle können Applikationen mit der HiPath4000 Nachrichten austauschen und somit können z.B. Call Control Funktionen umgesetzt werden.

Codierung:

Betrachtet werden muss, dass die gesamte PDU in zwei Schritten encodiert wird.

Zu Beginn wird die gefüllte Struktur nach der gewünschten Codierungsvorschrift encodiert. Im Falle dieser Diplomarbeit handelt es sich um die BER Codierungsvorschrift. Diesem erzeugten Oktettstrom wird der Rose-Header angefügt.

R.O.S.E. - Header

Mittels den Informationen im ROSE[23] Header lassen sich Response den Request PDU's zuordnen.

Im ROSE Header werden die Informationen zu den PDU's und dem Typ der übertragenen Struktur gespeichert. Der Header besteht aus vier Feldern, der Invoke ID, der Linked ID, dem Operation ID und dem Message Data.

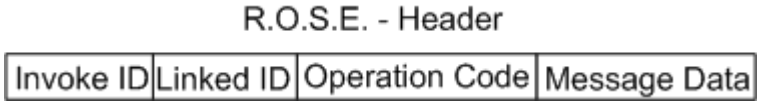


Abbildung 33: R.O.S.E. Header

In der obigen Abbildung ist der Aufbau des R.O.S.E. Headers dargestellt.

Der Invoke ID besitzt die Funktion eines Handlers, um Response Messages den Request Messages zuzuordnen.

Die Linked ID wird dann verwendet, wenn eine „Unteroperation" initialisiert wird. Dann ist die Linked ID die Invoke ID der Eltern – Operation.

[23] Remote Operations Service Element

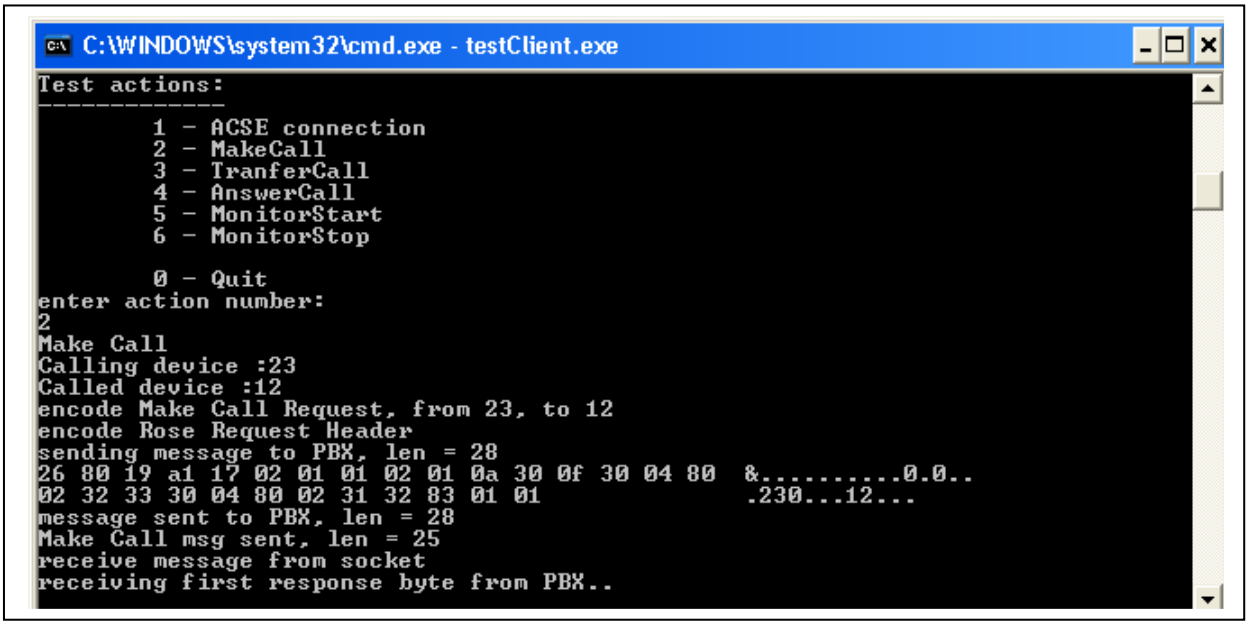
Der Operation Code gibt an, zu welcher Struktur die nachfolgenden Daten gehören.

Nachricht	Dez	Hex
Make Call	10	0A
Hold Call	9	09
Clear Call	4	04
Answer Call	2	02
Park Call	18	12
Join Call	223	0F
Alternate Call	1	01
Conference Call	6	06
Monitor Start	71	47
Monitor Stop	73	49
Snapshot Call	75	4B
Snapshot Device	74	4A
IO Register	340	01 54
Start Data Path	110	6E
Stop Data Path	111	6F
Data Path Resumed	118	76
Data Path Suspended	116	74
Send Data	112	70
Multicast Data	113	71
Send Broadcast Data	114	72
System Status	211	D3
System Register	207	CF

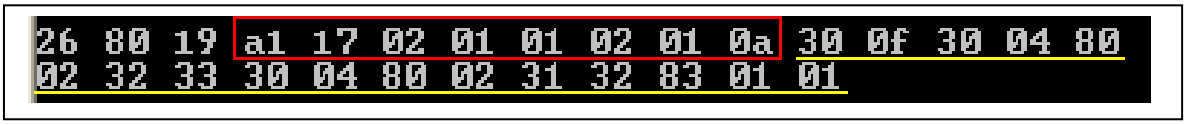
Tabelle 13: Operation Code in Dezimal und Hexadezimal

Die Tabelle 13 beinhaltet alle für die Verwendung des Treibers notwendigen Operation Codes. Dieser wird beim Empfänger decodiert, so dass das System die Art der Informationen der folgenden Nachricht zuordnen kann.

Als Beispiel wird eine Make-Call PDU mit den Parameter Calling Device = 23 und Called Device = 12 gefüllt.



Nachfolgend ist der R.O.S.E – Header aus diesem Beispiel, vergrößert dargestellt.



Der rot markierte Abschnitt in der oberen Abbildung beschreibt den R.O.S.E – Header.

In der folgenden Tabelle sind die Oktetts und deren Bedeutung aufgelistet.

Oktett	Bedeutung
A1	Request
17	Länge
02	Typ: Integer
01	Länge
01	Wert – Invoke ID
02	Typ: Integer
01	Länge
0A	Make-Call

Die gelb unterstrichenen Hexwerte entsprechen dem Message Data und werden nicht weiter betrachtet.

5 Umsetzung, Implementierung und Test

5.1 Einleitung

Als Grundlage für die Umsetzung diente ein bereits vorhandenes Projekt, wodurch schon eine gewisse Grundstruktur vorgegeben war, das bereits implementiert, getestet wurde und Anwendung findet. Einige der notwendigen PDU's sind bereits implementiert.

5.2 Umsetzung

In dem vorhandenen Projekt wurde eine DLL Datei erzeugt und von einer Applikation aufgerufen.
Dabei sind schon folgende Dateien angelegt und implementiert worden:

CstaApi.cpp	DLL Funktionen definiert + ID Management
CstaApi.h	DLL Funktionen deklariert + Csta Strukturen angelegt
Asn1Impl.cpp	Hilfsfunktionen die bei Encodings und Decodings genutzt werden
PbxDrvCstaH4000TestDlg.cpp/.h	Test-GUI zum aufrufen der Funktionen, Verbinden mit CAP- Server, mittels Buttons senden von Funktionen und empfangen
PbxDrvCstaH4000Test.cpp/.h	Erzeugung der GUI

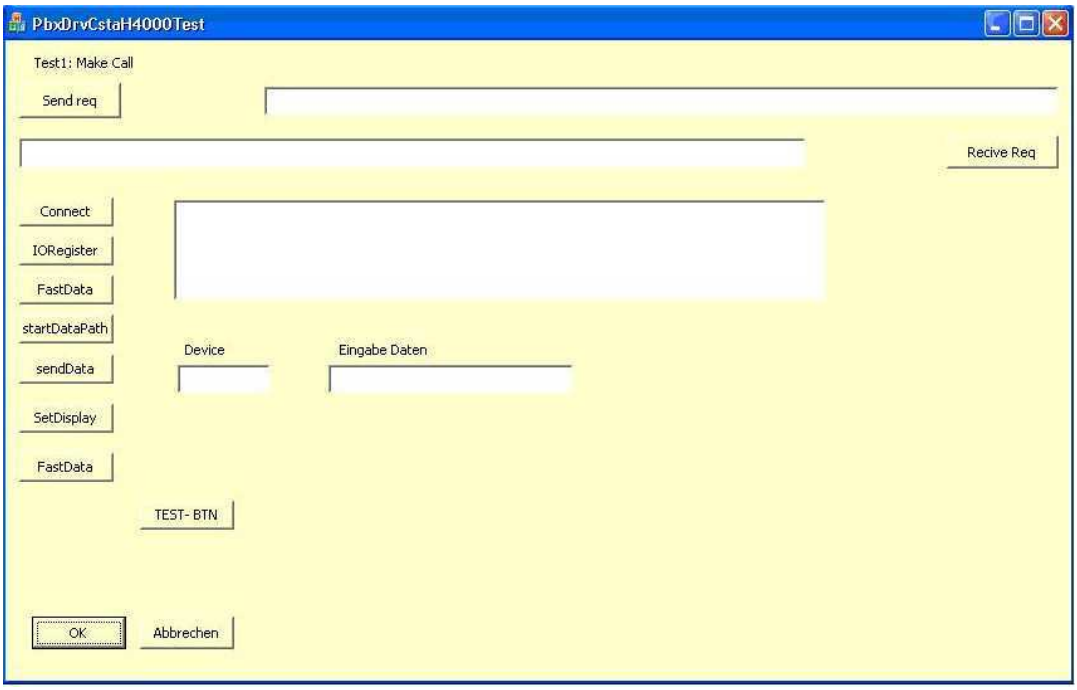


Abbildung 34: Programm zum Testen der Funktionen

Aus diesem Grund ist es nicht notwendig, eine eigene Umsetzung zu entwickeln, da diese Lösung sich schon in der Praxis bewährt hat.

Nachfolgend sind die bereits teilweise implementierten Strukturen aufgelistet:

- System Status Request
- Io Register Request
- Fast Data Response
- Make Call Request
- Monitor Start Request
- Set Display Request
- System Status Response
- Escape Request
- Data Path Response
- Send Data Request
- Button Press Request
- IO Register Response
- Fast Data Request
- Send Data Request
- Start Data Path Request
- Stop Data Path Request
- Button Press Event

Nachfolgend sollen die Hilfsfunktionen erklärt werden, die für die Umsetzung der Encoding und Decoding Regeln notwendig waren.

Funktionsaufruf	Input	Output	Bedeutung
_oss_encd_uint()	Struct ossGlobal *, char** pbuf, long*, char, long unsigned i	long	Umwandeln eines Integer Wertes in ein Hex–Wert
_oss_encd_nstr()	Struct ossGlobal* _g, char** pbuf, long*, char, char* c, long i	long	Umwandeln jedes Zeichen des String in den Hex-Wert
_oss_dec_iint()	struct ossGlobal*, char** pbuf,long* plong,long len	int	decoding of vartiable length int
_oss_dec_nstr()	struct ossGlobal *_g, char** pbuf,long*,long len, char* dst, long max, char	void	decoding of strings

Da es die Aufgabe des Treibers ist, alle ID's intern zu verwalten, müssen dieses durch ein Management organisiert werden.
Es sind alle zu beachtenden ID-Typen aufgelistet und deren Organisation beschrieben.
Zu Beginn soll der Invoke ID betrachtet werden, da es den Hauptbestandteil des ROSE Headers darstellt. Um deren Organisation verstehen zu können, muss kurz auf deren Verwendung eingegangen werden. Auf jedem Request antwortet das System mit einem Response, das die gleiche Invoke ID besitzt. Dadurch kann das System bei erhöhten Datenaufkommen die Response-Nachricht der Request-Nachricht zuverlässig zuordnen.
Im Treiber ist der Invoke ID eine globale Variable die mit 1 initialisiert wird. Nach jedem Senden eines Requestes wird diese um eins erhöht. Bei Erreichen des Wertes 100 wird der Invoke ID wieder zurück gesetzt. Dadurch wird erreicht, dass ein zuverlässiges Zuordnen der Responses möglich ist und dass nicht durch einen Designfehler mehrere Request den gleichen Invoke ID verwenden.

Funktionsweise des Treibers:

Die einzelnen Funktionen stehen als API zur Verfügung.
In der anknüpfenden Grafik ist der grundlegende Ablauf, welcher bei einem Aufruf einer Funktion stattfindet.

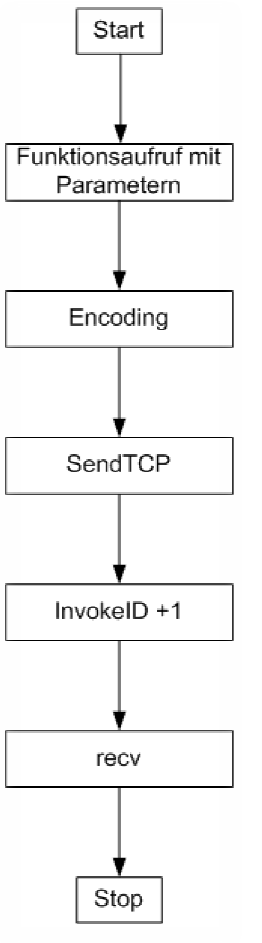


Abbildung 35: Allgemeiner Ablauf bei einem Funktionsaufruf

Zum besseren Verständnis wird der Ablauf für die Funktion „`sendMakeCallRequest()`“ durchlaufen. Der Funktion wird als Parameter die `CstaMakeCallRequest` – Struktur übergeben.

```
struct CstaMakeCallRequest {
   char* callingDevice;
   char* calledDirectoryNumber;
};
```

Diese besitzt `callingDevice` und `calledDirectoryNumber` als Elemente.
Bei Aufruf der Funktion beginnt die Encodierung von „hinten nach vorne“, die Information, die als letztes im Bitstrom steht, werden als erstes encodiert.

Nachfolgend ist der Programmausschnitt dargestellt. Die ausführlichen Kommentare befinden sich unter dem Quellcode.

```
void PDIAPI sendMakeCallRequest(CstaMakeCallRequest* csta) {
1     char* device = csta->callingDevice;
2     char* number = csta->calledDirectoryNumber;
3     char pdu[1024];
4     char* data=pdu+1024;
5     _oss_encd_nstr(0, &data, 0,0, number, 0);
6     int lenNumber = strlen(number);
7     _oss_encd_uint(0, &data, 0,0, lenNumber);
8     _oss_encd_uint(0, &data, 0,0, 0x80);
9     _oss_encd_uint(0, &data, 0,0, 2+lenNumber);
10    _oss_encd_uint(0, &data, 0,0, 0x30);
11    _oss_encd_nstr(0, &data, 0,0, device, 0);
12    int lenDevice = strlen(device);
13    _oss_encd_uint(0, &data, 0,0, lenDevice);
14    _oss_encd_uint(0, &data, 0,0, 0x80);
15    _oss_encd_uint(0, &data, 0,0, 2+lenDevice);
16    _oss_encd_uint(0, &data, 0,0, 0x30);
17    _oss_encd_uint(0, &data, 0,0, lenDevice+8+lenNumber);
18    _oss_encd_uint(0, &data, 0,0, 0x30);
19    sendRoseRequest(data,10+lenDevice+lenNumber,0x0A);
20    generateInvokeId();
}
```

3. Es wird ein Eindimensionales Array Feld der Größe 1024 angelegt. Dabei ist dies die maximale Größe, die die PDU annehmen kann.
4. Da die PDU von hinten nach vorne befüllt wird, zeigt ein Zeiger auf das letzte Feld der PDU.
5. Die calledDirectoryNumber ist vom Typ String und wird mittels der Hilfsfunktion _oss_encd_nstr() in eine Folge von Hex Werten umgewandelt. Dazu muss der Hilfsfunktion ein Pointer auf die nächste Stelle in der PDU und ein Zeiger auf den Beginn des Strings übergeben werden.

In Zeile 6 - 18 werden die BER Encoding Rules angewendet. Zu beachten ist, dass die PDU mit der Gesamtlänge und 0x00 abgeschlossen wird.

19. Mit sendRoseRequest wird die PDU an den SiemensCAP Server gesendet.

20. Zum Abschluss muss stets beim Senden von Request zur Unterscheidung die InvokeID um eins erhöht werden.

Die zu sendende PDU wird erst mit der Nummer der Called Directory und dann mit der calling Device befüllt. Da es dabei um Variablen handelt kann zu Beginn nicht abgeschätzt werden welchen Umfang sie einnehmen werden. Daher beginnt man bei der Zusammenstellung der PDU mit diesen Bestandteilen und baut darauf auf, siehe Abbildung 36 .

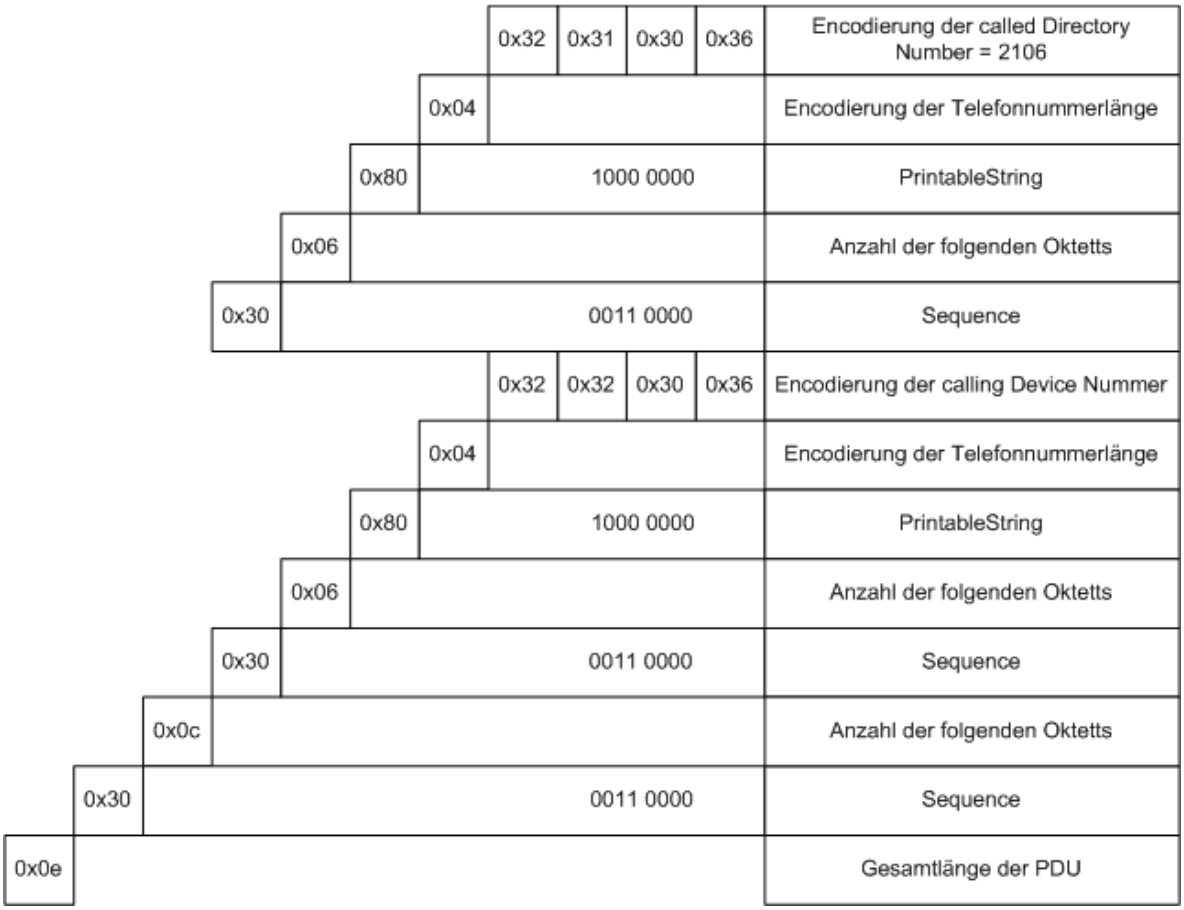


Abbildung 36: Make Call PDU

Manche Response geben als Rückgabe eine ID auf, die sich auf andere PDU's beziehen, wenn diese darauf aufbauen oder Informationen dazu liefern. Aus diesen Grund ist es notwendig, diese gewonnen ID´s besonders zu verwalten, um dann Events den richtigen Devices zuordnen zu können. Aus diesem Grund wird, wie nachfolgend in der Grafik zu erkennen ist, eine Tabelle angelegt, in der alle notwendigen Informationen global gespeichert sind.

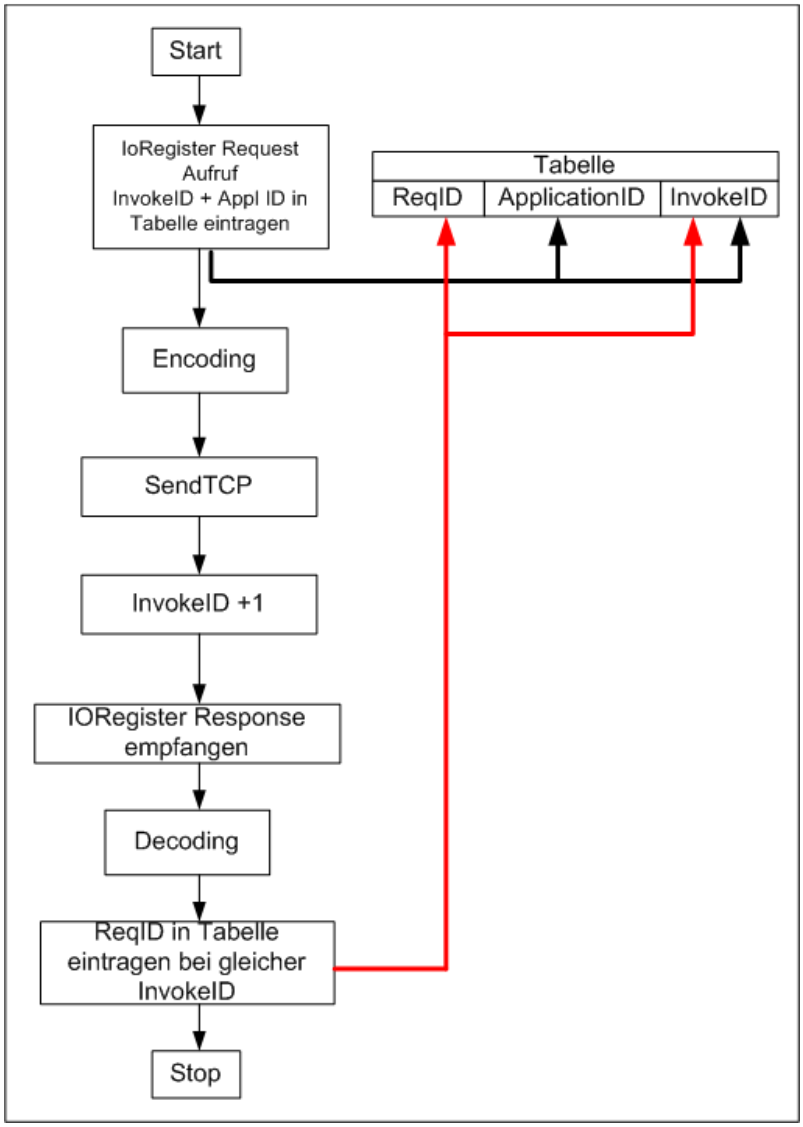


Abbildung 37 Ablauf bei der Funktion IoRegisterRequest()

Zur Verdeutlichung wird am Beispiel der PDU „sendCstaIoRegisterRequest()" der Ablauf erklärt werden.

Dieser Funktion wird als Paramter die „CstaIoRegisterRequest" Struktur übergeben.

Diese Struktur besitzt zum einen den Manufacturer und zum anderen die ApplicationID als Elemente. Beide Elemente können gefüllt werden. Für den derzeitigen Systemaufbau muss als Manufacturer „Siemens CAP" und als ApplicationID „559" eingetragen werden.

Für diese Funktion ist eine angepasste Tabelle notwendig. In der ersten Spalte werden die ReqID, in der zweiten die ApplicationID und in der dritten die InvokeID eingetragen.

Nach dem Aufrufen der sendCstaIoRegisterRequest() – Funktion wird die Tabelle mit der Application und der zum Request passenden InvokeID gefüllt.
Danach folgt der normale Ablauf, wie bereits oben beschrieben.
Wird darauf ein IoRegisterResponse empfangen, ist der erste Schritt, diese zu decodieren. Nachdem alle notwendigen Parameter decodiert sind, wird die oben erwähnte Tabelle nach der empfangen Invoke ID durchsucht. Bei Erfolg wird zur ApplicationID auch die ReqID eingetragen. Ist der InvokeID nicht in der Tabelle enthalten, wird die Response - Nachricht verworfen.
Durch dieses System ist es möglich, mehrere Applikationen zu verwalten.
Nach dem gleichen Prinzip verwaltet man die ID‘s der Data Paths.

Anschließend soll der Ablauf, wenn man die Funktion IoRegisterCancel aufruft, betrachtet werden. Beim Senden dieser Funktion wird die Application von der HiPath4000 abgemeldet. Dazu muss die dazugehörige ReqID als Parameter mitgesendet werden.
Jede ApplicationID kann nur eine ReqID zugewiesen werden. Die Application ID ist in den meisten Fällen eher bekannt als die ReqID. Aus diesem Grund wird über die ApplicationID auf die ReqID geschlossen. Der Ablaufplan ist in folgender Abbildung dargestellt. Nach Senden der PDU wird diese Zeile gelöscht.

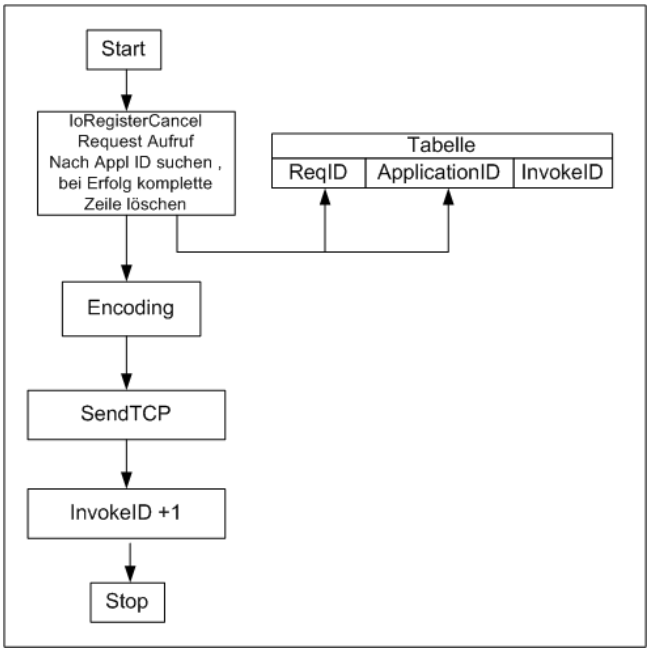


Abbildung 38: Ablauf der Funktion IoRegisterCancelRequest

Beim Empfangen einer PDU wird diese durch die Funktion recvTcp() weiterverarbeitet. Diese Funktion regelt außerdem den Fall, dass der interne Empfangspuffer mehrere PDU’s empfangen kann.

Dazu wird der komplette Empfangspuffer erst gesichert und dann an der 2. Position der Nachricht, die Gesamtlänge der PDU's in eine lokale Variable gespeichert. Sollte die lokale Variable sehr viel kleiner sein als die gesamte Länge der gesicherten Nachricht, müssen mehrere PDU's empfangen worden sein.
Wurde eine komplette PDU empfangen, wird mit der Funktion recvLayer4() geklärt, ob es sich bei der Nachricht vom Typ Request oder Response handelt.

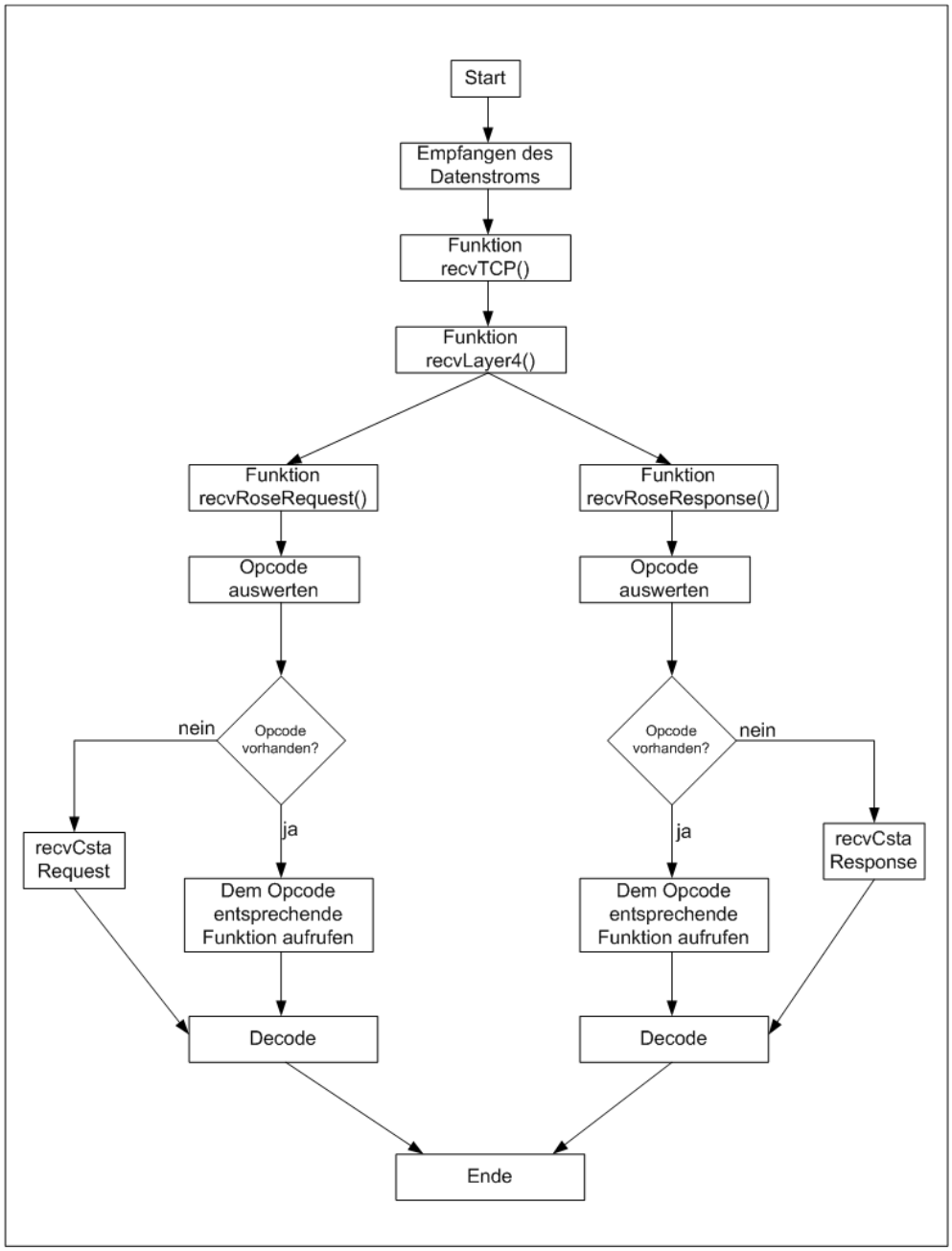


Abbildung 39: Allgemeiner Ablauf beim Empfangen eines Bitstroms

In der oberen Abbildung ist der komplette Ablauf für das Empfangen einer Nachricht dargestellt. Der Ablauf für recvRoseRequest() und recvRoseResponse verläuft fast identisch. Der

Unterschied besteht nur darin, welcher Wert im Opcode steht. Daraufhin wird die entsprechende Funkionen aufgerufen.
Das Abfragen des Opcodes realisiert man mittels einer Case Anweisung. Als Default wird die Funktion recvCstaResponse bzw. recvCstaRequest aufgerufen. Mit der aufgerufenen Funktion beginnt die Decodierung und alle wichtigen Informationen werden in der dazu passenden Struktur gespeichert.

Implementierte Funktionen sind anschließend aufgelistet.

- sendCstaSystemStatusRequest
- sendCstaIoRegisterRequest
- sendCstaIoRegisterCancelRequest
- sendFastDataResponse
- sendMakeCallRequest
- sendMonitorStartRequest
- sendMonitorStopRequest
- sendSetDisplayRequest
- sendStartDataPathResponse
- sendStartDataPathRequest
- sendStopDataPathRequest
- sendSendDataRequest
- sendButtonPressRequest
- sendSetDisplayRequest
- sendFastDataRequest
- sendAlternateCallRequest
- sendConferenceCallRequest
- sendHoldCallRequest
- sendClearConnectionRequest
- sendAnswerCallRequest
- sendParkCallrequest
- sendSnapshotDeviceRequest
- recvCstaIoRegisterResponse
- recvFastDataRequest
- recvSendDataRequest
- recvStartDataPathRequest
- recvStopDataPathRequest
- recvButtonPressEvent

- recvStartDataPathResponse
- recvMonitorStartResponse
- recvSnapshotDeviceResponse
- recvCstaSnapshotDeviceDataRequest

5.3 Test

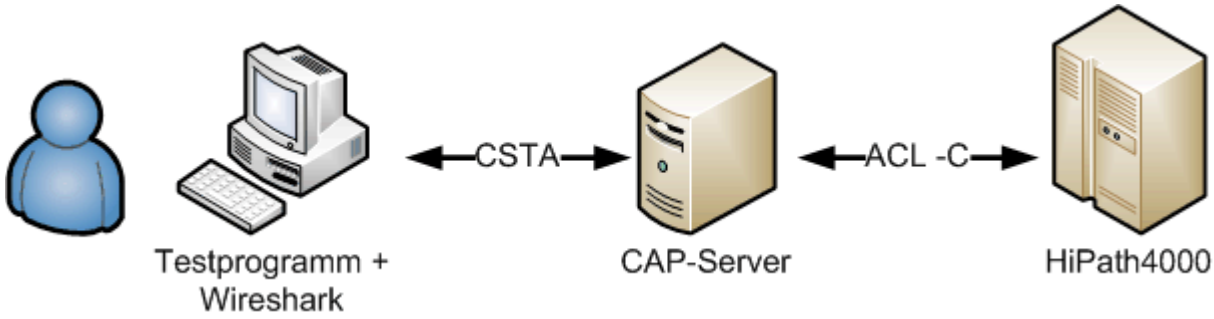


Abbildung 40: Testabbildung

Die obere Abbildung stellt das Testsystem dar. Auf dem Rechner mit dem Testprogramm wird die Software Wireshark verwendet, um den Datenaustausch zwischen dem Testprogramm und dem Cap Server aufzuzeichnen. Im nachhinein werden die gesendeten Nachrichten ausgewertet. Über die Eingabe befüllt sich die PDU mit Testinformationen. Syntaktische Fehler werden bereits durch den Compiler angezeigt und können daraufhin behoben werden.

Die Designfehler können erst durch das Auswerten der gesendeten PDU's erkannt und beseitigt werden. Ein Hinweis auf einen Designfehler wäre, wenn die HiPath4000 ein fehlerhaftes Response an die Applikation sendet. Um dies zu erkennen, wird mittels Wireshark die gesendete PDU mit einer bereits erzeugten PDU verglichen. Die Vergleichs-PDU wird mit Hilfe des Programmes CSTA3Host von Siemens erzeugt. Ein weiterer Hinweispunkt auf ein Designfehler wäre, wenn die HiPath4000 in der Response-Nachricht einen Fehler sendet. Durch den Fehlercode ist es möglich die Fehlerquelle einzugrenzen. Ein Fehler besitz sowohl einen Fehlertyp als auch einen Fehlerwert. Beispielsweise gehört der Operation Errors gehört zu den Fehlertypen und generic oder Invalid ID zu den Fehlerwerten. Durch diese Struktur wird das Auffinden der Fehlerursache und deren Beseitigung erleichtert.

In der folgenden Abbildung wird eine kurze Übersicht über die Error Struktur gegeben.

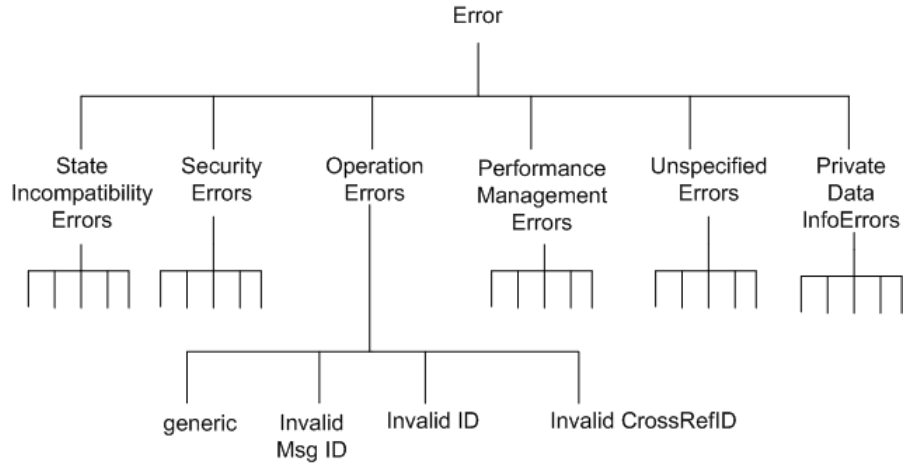


Abbildung 41: Errors

Für die genauere Fehlerbeschreibung muss der Standard herangezogen werden. Der Standard kann bei der ecma-International kostenlos heruntergeladen werden (http://www.ecma-international.org/activities/Communications/TG11/cstaIII.htm).

Wie ein Error Response aussehen kann ist am Beispiel eines CSTA Universal Error verdeutlicht.

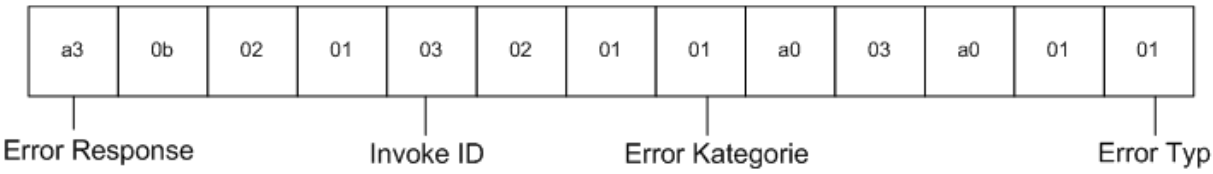


Abbildung 42: Universal Error

Der Hexadezimal Wert „A3“ steht dafür das die PDU einen Fehler beschreibt, da womöglich ein gesendetes Request nicht verarbeitet werden konnte.
Wichtige Informationen welche PDU den Fehler verursacht hat gibt die Invoke ID an.
Das Feld Error Kategorie gibt an ob der Fehler im Bereich State Incompatibility, Security oder Operation liegt. In diesem Beispiel handelt es sich um einen Operation Error. der Error Typ spezifiziert dann den Fehler genauer. Der in diesem Beispiel erhaltene Wert 0x01 steht für einen generic Error. Welcher Hexadezimal Wert für welchen Fehler steht ist im Standard nachzulesen.

6 Zusammenfassung

6.1 Ergebnisse

Am Anfang der Diplomarbeit stand die Recherche, nach der besten Vorgehensweise dieses Projekt umzusetzen. Es wurde geprüft ob eine Anschaffung eines ASN.1 Compilers sinnvoll wäre. Dazu gehörte auch das Einarbeiten in die diversen Tools um deren Stärken und Schwächen kennen zu lernen. Das Resultat der Einarbeitung war das 2 Tools in Frage kommen würden, aber durch die enormen Kosten, wurde sich gegen eine Anschaffung entschieden.
Daraufhin erfolgte die Umsetzung des CSTA Standards, vorliegend in ASN.1 Syntax, es wurden nur die für notwendig betrachteten PDU's umgesetzt. Als unterstützendes Tool wurde die Applikation CSTA3Host.exe, von Siemens, verwendet.
Beim Implementieren der Send Data Request PDU war ein Problem aufgetreten. Die Option die es ermöglicht, dass der Anzeigetext auf dem Device stehen bleibt ließ sich nicht realisieren. Auch nach langer Phase der Recherche und des Probierens war es nicht möglich dieses Problem zu beheben. Auch eine Anfrage an den Entwickler der HiPath4000 verlief erfolglos.
Es wurden alle gewünschten Strukturen erfolgreich umgesetzt und getestet.
Das praktische Umsetzen der Diplomarbeit erfolgte in einem bereits vorhandenen Projekt.
Die schriftliche Arbeit beginnt mit einer Einführung und anschließend wurden die Grundlagen ASN.1, CSTA und Teile aus den UML Diagrammen ausführlich behandelt. Dem folgt dann die Ergebnisse aus der Recherche. Danach beginnt die Beschreibung des eigenen Anteils an der Arbeit, dies umfasst die Modellierung, die Umsetzung und den Test. Die Diplomarbeit wird mit Fazit und Ausblick abgerundet.

Im Rahmen dieser Arbeit ist ein funktionierender CSTA- Treiber entstanden. Die Encodierung und Decodierung der wichtigsten PDU ist Implementiert. Der Verbindungsaufbau und -abbau wird vom Treiber realisiert. Es stehen Funktionen zur Verfügung um eine Call Control zu verwirklichen. Es wurden Funktionen implementiert die es ermöglichen an Devices Daten zu schicken, die jene über das Display ausgeben. Außerdem werden die für das senden von Daten notwendigen ID's vom Treiber verwaltet.

6.2 Ausblick

In diesem Kapitel möchte der Autor mögliche Ideen zur weiteren Entwicklung des Treibers geben.

Im CSTA-Treiber sind nur einzelne ausgewählte Events implementiert wie zum Beispiel Button Press Event. Daher wird vorgeschlagen zu Untersuchen ob es notwendig wäre zur besseren Verwendung weitere Events zu realisieren.

Des Weiteren wäre sinnvoll durch weiteres Recherchieren den Fehler mit Send Data, in Kapitel 6.1 beschrieben, zu lösen. Eine Möglichkeit wäre, die Option des stehenden Textes, mittels Threads im Treiber zu realisieren. Dabei sollte aber beachtet werden auf Hinblick auf eine mögliche Portabilität eine Thread Bibliothek zu wählen die sowohl vom gcc als auch vom gnu Compiler umgesetzt werden kann.

Der gesamte Code sollte auf eine mögliche Portation auf ein Linux System geprüft werden.

Außerdem muss geprüft werden on der Treiber sich in das vorhandene Zeiterfassungssystem integrieren lässt, ansonsten müssten noch Veränderungen vorgenommen werden.

Da während der Diplomarbeit es nicht möglich war, einen Laufzeittest mit dem Treiber durchzuführen. Wäre dies für den nächsten Schritt anzuraten, dazu müsste eine passende Testapplikation entwickelt werden.

Anlagen

Anlage A: Der Quelltext der CSTA - DLL

Die Cpp-Datei des CSTA - Treibers

```
/*************************************************************************
** Filename:      CstaApi.cpp                                           **
** Version        1.0                                                   **
** Autor:         Markus Franke, Rico Thomanek                          **
** Inhalt:        Funktionen zur Realisierung der Decodierung            **
**                und Encoderiung                                       **
*************************************************************************/
#include <Afxsock.h>
#include <process.h>
#include <stdio.h>
#include <time.h>
#include "syserr.h"
#include "stdafx.h"


CAsyncSocket g_socket;


#include "CstaApi.h"
#include "Asn1Impl.cpp"


//global definierte Variblen
   char g_Device[8];
   char g_Manufacturer[120];
   char g_ApplicationId[6];
   char g_char_fastData[1024];
   int g_length_fastData;
   int g_int_i;
   int g_index_device;
   MSG msg;
   int g_lineDelete;
   int g_ioCrossRefId;
   int arrayCstaMonitorRefId[2][20];
   int zeileMonitorRefId = 0;

   int arrayCstaIoRegisterReqId[3][5];
   int zeileIoRegisterReqId = 0;

   int arrayCstaIoCrossRefId[3][30];
   int zeileIoCrossRefId = 0;

   int g_InvokeId = 1;
```

```
void sendLayer4(char* data,int len) {
   _oss_encd_uint(0, &data, 0,0, len);
   _oss_encd_uint(0, &data, 0,0, 0x80);
   _oss_encd_uint(0, &data, 0,0, 0x26);
   sendTcp(data,3+len);
}

// A1 2A 02 01 02 02 02 01 12
int sendRoseRequest(char* data,int len,int op,int invokeId) {
   char* base=data;
   //opValue
   int vintlen=_oss_encd_int(0, &data, 0,0, op);
   _oss_encd_uint(0, &data, 0,0, vintlen);
   _oss_encd_uint(0, &data, 0,0, 0x02);
   // invokeId
   _oss_encd_uint(0, &data, 0,0, invokeId);
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x02);
   // Request
   int roseLen=(int)base-(int)data;
   _oss_encd_uint(0, &data, 0,0, roseLen+len);
   _oss_encd_uint(0, &data, 0,0, 0xa1);
   sendLayer4(data,2+roseLen+len);
   return invokeId;
}

void sendRoseResponse(char* data,int len,int op,int invokeId) {
   char* base=data;
   //opValue
   int vintlen=_oss_encd_int(0, &data, 0,0, op);
   _oss_encd_uint(0, &data, 0,0, vintlen);
   _oss_encd_uint(0, &data, 0,0, 0x02);
   // invokeId
   _oss_encd_uint(0, &data, 0,0, invokeId);
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x02);
   // Request
   int roseLen=(int)base-(int)data;
   _oss_encd_uint(0, &data, 0,0, roseLen+len);
   _oss_encd_uint(0, &data, 0,0, 0xa2);
   sendLayer4(data,2+roseLen+len);
}


//a1 0c 02 01 01 02 02 00 d3 30 03 0a 01 00
void PDIAPI sendCstaSystemStatusRequest(CstaSystemStatusRequest* csta){
   char pdu[1024];
   char* data=pdu+1024;
   char* base=data;
   int system_status_arg = csta->SystemStatusArg;
   int system_status = csta->SystemStatus;
   int invokeId=1;
   int vintlen;

   //SystemStatusArg
   vintlen=_oss_encd_uint(0, &data, 0,0, system_status_arg);
   _oss_encd_uint(0, &data, 0,0, vintlen);
   _oss_encd_uint(0, &data, 0,0, 0x0a);
   _oss_encd_uint(0, &data, 0,0, 2+vintlen);
   _oss_encd_uint(0, &data, 0,0, 0x30);
   //SystemStatus
   vintlen=_oss_encd_int(0, &data, 0,0, system_status);
```

```
    _oss_encd_uint(0, &data, 0,0, vintlen);
    _oss_encd_uint(0, &data, 0,0, 0x02);
    //InvokeID
    vintlen=_oss_encd_int(0, &data, 0,0, g_InvokeId);
    _oss_encd_uint(0, &data, 0,0, vintlen);
    _oss_encd_uint(0, &data, 0,0, 0x02);
    // Request
    _oss_encd_uint(0, &data, 0,0, (int)base-(int)data);
    _oss_encd_uint(0, &data, 0,0, 0xa1);
    //Länge PDU
    _oss_encd_uint(0, &data, 0,0, (int)base-(int)data);
    _oss_encd_uint(0, &data, 0,0, 0x00);
    sendTcp(data,3+(int)base-(int)data);
    generateInvokeId();
}


//30 10 30 05 80 03 31 30 30 30 07 80 05 2A 34 32 31 23
void PDIAPI sendMakeCallRequest(CstaMakeCallRequest* csta) {
    char* device = csta->callingDevice;
    char* number = csta->calledDirectoryNumber;
    char pdu[1024];
    char* data=pdu+1024;
    _oss_encd_nstr(0, &data, 0,0, number, 0);
    int lenNumber = strlen(number);
    _oss_encd_uint(0, &data, 0,0, lenNumber);
    _oss_encd_uint(0, &data, 0,0, 0x80);
    _oss_encd_uint(0, &data, 0,0, 2+lenNumber);
    _oss_encd_uint(0, &data, 0,0, 0x30);
    _oss_encd_nstr(0, &data, 0,0, device, 0);
    int lenDevice = strlen(device);
    _oss_encd_uint(0, &data, 0,0, lenDevice);
    _oss_encd_uint(0, &data, 0,0, 0x80);
    _oss_encd_uint(0, &data, 0,0, 2+lenDevice);
    _oss_encd_uint(0, &data, 0,0, 0x30);
    _oss_encd_uint(0, &data, 0,0, lenDevice+8+lenNumber);
    _oss_encd_uint(0, &data, 0,0, 0x30);
    sendRoseRequest(data,10+lenDevice+lenNumber,0x0A);
}


//a1 19 02 01 13 02 01 02 30 11 6b 0f 30 0d 80 01 29<-ConnId a1 08 30 06 80
04 32 31 32 36
void PDIAPI sendAnswerCallRequest(CstaAnswerCallRequest *csta){


    char pdu[1024];
    char* data=pdu+1024;
    int opcodeId = 0x02;
    int invokeId = 0x40;


    int L1 = _oss_encd_nstr(0,&data, 0,0, csta->device,0);
    _oss_encd_uint(0,&data,0,0,L1);
    _oss_encd_uint(0,&data,0,0,0x80);
    _oss_encd_uint(0,&data,0,0,L1+2);
    _oss_encd_uint(0,&data,0,0,0x30);
    _oss_encd_uint(0,&data,0,0,L1+4);
    _oss_encd_uint(0,&data,0,0,0xA1);

    _oss_encd_uint(0,&data,0,0,csta->connId);
    _oss_encd_uint(0,&data,0,0,0x01);
```

```
   _oss_encd_uint(0,&data,0,0,0x80);

   _oss_encd_uint(0,&data,0,0,L1+9);
   _oss_encd_uint(0,&data,0,0,0x30);
   _oss_encd_uint(0,&data,0,0,L1+11);

   _oss_encd_uint(0,&data,0,0,0x6B);
   _oss_encd_uint(0,&data,0,0,L1+13);
   _oss_encd_uint(0,&data,0,0,0x30);

   //a1 19 02 01 13 02 01 02
   _oss_encd_uint(0,&data,0,0,opcodeId);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);

   _oss_encd_uint(0,&data,0,0,g_InvokeId);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);

   _oss_encd_uint(0,&data,0,0,L1+21);
   _oss_encd_uint(0,&data,0,0,0xA1);

   _oss_encd_uint(0,&data,0,0,L1+23);
   _oss_encd_uint(0,&data,0,0,0x00);
   sendTcp(data,L1+25);
   generateInvokeId();

}

//30 10 E1 0E 06 06 2B 0C 02 88 53 0F 04 04 1A 04 01 C0
void PDIAPI sendEscapeRequest() {
   char pdu[1024];
   int len=scanHex("30 10 E1 0E 06 06 2B 0C 02 88 53 0F 04 04 1A 04 01 C0
",pdu);
   sendRoseRequest(pdu,len,0x33);

}


void PDIAPI sendAARQ() {
   char pdu[1024];
   int len=scanHex("60 31 A1 07 06 05 2B 0C 00 81 5A 8A 02 06 80 AC 15 A2
13 A0 11 A0 0F 04 06 41 4D 48 4F 53 54 04 05 37 37 37 37 37 BE 0B 28 09 A0
07 A0 05 03 03 00 10 00 ",pdu);
   sendLayer4(pdu,len);
}


//a1 2a 02 01 03 02 01 6e 30 22 30 06 80 04 (32 31 32 36)<--CallId 7e 18 a1
16 bc 14 04 0b 53 69 65 6d 65 6e 73 20 43 41 50 b0 05 8203 35 35 39
void sendStartDataPathRequest( CstaFastDataRequest2 *csta){ //<--aufgerufen
nach FastData
char   pdu[1024];
char   tmp[1024];
char*  data= pdu+1024;
char*  base=data;
char*  applicationId=csta->applicationId;
char*  manufacturer=csta->manufacturer;


   //umwandeln Hex-String in Integer
   int nResult;
```

```
    sscanf(csta->device, "%x", &nResult);

    // Tabelle mit invokeID und Application ID anlegen
    arrayCstaIoCrossRefId[1][zeileIoCrossRefId] = nResult;
    arrayCstaIoCrossRefId[2][zeileIoCrossRefId] = g_InvokeId;
    zeileIoCrossRefId++;

        if(zeileIoCrossRefId == 29) zeileIoCrossRefId=0;

    //Zeiger um eins erhöhen bei 20 wieder von vorn beginnen


// Private data
_oss_encd_nstr(0, &data, 0,0, applicationId, 0);
    int L1 = strlen(csta->applicationId);
    _oss_encd_uint(0, &data, 0,0, L1);
    _oss_encd_uint(0, &data, 0,0, 0x82);
    _oss_encd_uint(0, &data, 0,0, 2+L1);
    _oss_encd_uint(0, &data, 0,0, 0xb0);
    int L2 = strlen(csta->manufacturer);
    _oss_encd_nstr(0, &data, 0,0, manufacturer, 0);
    _oss_encd_uint(0, &data, 0,0, L2);
    _oss_encd_uint(0, &data, 0,0, 0x04);
    _oss_encd_uint(0, &data, 0,0, L1+6+L2);
    _oss_encd_uint(0, &data, 0,0, 0xbc);
    _oss_encd_uint(0, &data, 0,0, L1+L2+8);
    _oss_encd_uint(0, &data, 0,0, 0xa1);
    _oss_encd_uint(0, &data, 0,0, L1+L2+10);
    _oss_encd_uint(0, &data, 0,0, 0x7e);


// Encodierung von CallId
int vintlen=_oss_encd_nstr(0,&data,0,0,csta->device,0);
    _oss_encd_uint(0,&data,0,0,vintlen);
    _oss_encd_uint(0,&data,0,0,0x80);
    _oss_encd_uint(0,&data,0,0,vintlen+2);

    _oss_encd_uint(0,&data,0,0,0x30);
    _oss_encd_uint(0,&data,0,0,vintlen+L1+L2+16);
    _oss_encd_uint(0,&data,0,0,0x30);

    //a1 2a 02 01 03 02 01 6e

    _oss_encd_uint(0,&data,0,0,0x6E);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,g_InvokeId);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,vintlen+L1+L2+24);
    _oss_encd_uint(0,&data,0,0,0xA1);

    /*
    int len =scanHex("a1 2a 02 01 03 02 01 6e",tmp);
    data-=len;
    memcpy(data,tmp,len);
    */
    _oss_encd_uint(0,&data,0,0,vintlen+L1+L2+26);
    _oss_encd_uint(0,&data,0,0,0x00);

    sendTcp(data,vintlen+L1+L2+28);
    generateInvokeId();
```

```
}

//a1 0d 02 01 09 02 01 6f 30 05 a0 03 04 01 01
void PDIAPI sendStopDataPathRequest(CstaStopDataPathRequest *csta){
char   pdu[1024];
char*  data= pdu+1024;

int crossRefId;
int nResult;
sscanf(csta->device, "%x", &nResult);

   //nach DeviceId suchen und dazugehörigen ReqId heraussuchen um diesen
dann in pdu zu packen

           for(int i = 0;i < 29; i++) {
               if(arrayCstaIoCrossRefId[1][i]== nResult)
               {
                   crossRefId = arrayCstaIoCrossRefId[0][i];
                   arrayCstaIoCrossRefId[0][i] = 0;
                   arrayCstaIoCrossRefId[1][i] = 0;
                   arrayCstaIoCrossRefId[2][i] = 0;
                   i = 30;
               }
           }



   int vintlen=_oss_encd_int(0, &data, 0,0, crossRefId);
   _oss_encd_uint(0, &data, 0,0, vintlen);
   _oss_encd_uint(0, &data, 0,0, 0x04);

   _oss_encd_uint(0, &data, 0,0,vintlen+2);
   _oss_encd_uint(0, &data, 0,0, 0xA0);
   _oss_encd_uint(0, &data, 0,0,vintlen+4);
   _oss_encd_uint(0, &data, 0,0, 0x30);
   _oss_encd_uint(0, &data, 0,0, 0x6F); // Opcode
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x02);

   int invokeLen = _oss_encd_uint(0, &data, 0,0, g_InvokeId);
   _oss_encd_uint(0, &data, 0,0, invokeLen);
   _oss_encd_uint(0, &data, 0,0, 0x02);

   _oss_encd_uint(0, &data, 0,0,vintlen + invokeLen + 11);
   _oss_encd_uint(0, &data, 0,0, 0xA1);

   _oss_encd_uint(0, &data, 0,0,vintlen + invokeLen + 13);
   _oss_encd_uint(0, &data, 0,0, 0x00);

   sendTcp(data,vintlen + invokeLen + 15);
   generateInvokeId();
}


//a2 0f 02 01 02 30 0a 02 01 6e 30 05 a0 03 04 01 01 -- Hipath 4000
//A2 10 02 02 00 08 30 0A 02 01 6E 30 05 A1 03 04 01 06 -- Hipath 3000
void PDIAPI sendStartDataPathResponse(int invokeId,int crossRefId) {
   char pdu[1024];
   char* data=pdu+1024;
   char* base=data;
   //
   int vintlen=_oss_encd_int(0, &data, 0,0, crossRefId);
```

```
    _oss_encd_uint(0, &data, 0,0, vintlen);
    _oss_encd_uint(0, &data, 0,0, 0x04);
    _oss_encd_uint(0, &data, 0,0, 2+vintlen);
    _oss_encd_uint(0, &data, 0,0, 0xa1);
    _oss_encd_uint(0, &data, 0,0, 4+vintlen);
    _oss_encd_uint(0, &data, 0,0, 0x30);
    //opValue
    _oss_encd_uint(0, &data, 0,0, 0x6e);
    _oss_encd_uint(0, &data, 0,0, 0x01);
    _oss_encd_uint(0, &data, 0,0, 0x02);
    // 0x30
    _oss_encd_uint(0, &data, 0,0, 9+vintlen);
    _oss_encd_uint(0, &data, 0,0, 0x30);
    // invokeId
    vintlen=_oss_encd_int(0, &data, 0,0, invokeId);
    _oss_encd_uint(0, &data, 0,0, vintlen);
    _oss_encd_uint(0, &data, 0,0, 0x02);
    // Request
    _oss_encd_uint(0, &data, 0,0, (int)base-(int)data);
    _oss_encd_uint(0, &data, 0,0, 0xa2);
    sendLayer4(data,(int)base-(int)data);
}



//a2 0b 02 01 01 30 06 02 02 00 d3 05 00 -- Hipath 4000
//A2 0C 02 02 00 01 30 06 02 02 00 D3 05 00 //real
//A2 0C 02 02 00 01 02 02 00 D3 30 02 05 00 //falsch
void PDIAPI sendSystemStatusResponse(int invokeId) {
    char tmp[1024];
    char pdu[1024];
    char* data=pdu+1024;
    char* base=data;
    //int len=scanHex("30 02 05 00",pdu);
    //sendRoseResponse(pdu,len,0xd3,invokeId);

    int len=scanHex("30 06 02 02 00 D3 05 00",tmp);
    data-=len;
    memcpy(data,tmp,len);

    // invokeId
    int vintlen=_oss_encd_int(0, &data, 0,0, invokeId);
    _oss_encd_uint(0, &data, 0,0, vintlen);
    _oss_encd_uint(0, &data, 0,0, 0x02);
    // Response
    //int roseLen=11;
    int roseLen=(int)base-(int)data;
    _oss_encd_uint(0, &data, 0,0, roseLen);
    _oss_encd_uint(0, &data, 0,0, 0xa2);

    //Gesamtlänge
    vintlen=_oss_encd_int(0, &data, 0,0, 2+roseLen);
    _oss_encd_uint(0, &data, 0,0, 0x00);


    //sendLayer4(data,2+roseLen); für Hipath 4000 nicht notwendig
    sendTcp(data,1+vintlen+2+roseLen);
    //sendTcp(data,2+roseLen);
}

//a1 27 02 01 01 02 02 01 54 30 1e 30 02 80 00 7e 18 a1 16 bc 14 04 0b 53
69 65 6d 65 6e 73 20 43 41 50 b0 05 82 03 35 35 39
```

```
void PDIAPI sendCstaIoRegisterRequest(CstaIoRegisterRequest* csta) {
   char pdu[1024];
   char* data=pdu+1024;
   char* base=data;
   char* applicationId=csta->applicationId;
   char* manufacturer=csta->manufacturer;
   //int invokeId=1;
   int vintlen;


   //umwandeln Hex-String in Integer
   int nResult;
   sscanf(applicationId, "%x", &nResult);

   // Tabelle mit invokeID und Application ID anlegen
   //  int arrayCstaIoRegisterReqId[2]<-spalte[5]<-zeile;
   //  int zeileIoregisterReqId = 0;
   arrayCstaIoRegisterReqId[1][zeileIoRegisterReqId] = nResult;
   arrayCstaIoRegisterReqId[2][zeileIoRegisterReqId] = g_InvokeId;
   zeileIoRegisterReqId++;

       if(zeileIoRegisterReqId == 4) zeileIoRegisterReqId=0;

   //zeiger um eins erhöhen bei 20 wieder von vorn beginnen


   _oss_encd_nstr(0, &data, 0,0, applicationId, 0);

   int lenApplicationId =strlen(applicationId);
   _oss_encd_uint(0, &data, 0,0,lenApplicationId );
   _oss_encd_uint(0, &data, 0,0, 0x82);
   _oss_encd_uint(0, &data, 0,0, 2+lenApplicationId);
   _oss_encd_uint(0, &data, 0,0, 0xb0);
   _oss_encd_nstr(0, &data, 0,0, manufacturer, 0);
   int lenManufacturer = strlen(manufacturer);
   _oss_encd_uint(0, &data, 0,0,lenManufacturer);
   _oss_encd_uint(0, &data, 0,0, 0x04);
   _oss_encd_uint(0, &data, 0,0, lenApplicationId+6+lenManufacturer);
   _oss_encd_uint(0, &data, 0,0, 0xbc);
   _oss_encd_uint(0, &data, 0,0, lenApplicationId+lenManufacturer+8);
   _oss_encd_uint(0, &data, 0,0, 0xa1);
   _oss_encd_uint(0, &data, 0,0, lenApplicationId+lenManufacturer+10);
   _oss_encd_uint(0, &data, 0,0, 0x7e);
   _oss_encd_uint(0, &data, 0,0, 0x00);
   _oss_encd_uint(0, &data, 0,0, 0x80);
   _oss_encd_uint(0, &data, 0,0, 0x02);
   _oss_encd_uint(0, &data, 0,0, 0x30);
   _oss_encd_uint(0, &data, 0,0, lenApplicationId+lenManufacturer+16);
   _oss_encd_uint(0, &data, 0,0, 0x30);
   _oss_encd_uint(0, &data, 0,0, 0x54);
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x02);
   _oss_encd_uint(0, &data, 0,0, 0x02);
   _oss_encd_uint(0, &data, 0,0, g_InvokeId);
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x02);
   vintlen = 1;
   _oss_encd_uint(0, &data, 0,0,
vintlen+lenApplicationId+lenManufacturer+24);
   _oss_encd_uint(0, &data, 0,0, 0xa1);
```

```
   _oss_encd_int(0, &data, 0,0,
vintlen+lenApplicationId+lenManufacturer+26);
   _oss_encd_uint(0, &data, 0,0, 0x00);
   vintlen=(int)base-(int)data;
   sendTcp(data,(int)base-(int)data);
   generateInvokeId();
}

//a1 0c 02 01 01 02 02 01 56 30 03 04 01 01
void PDIAPI sendCstaIoRegisterCancelRequest (CstaIoRegisterCancelRequest*
csta){
   char pdu[1024];
   char* data=pdu+1024;
   int RegisterReqId;
   int nResult;
   sscanf(csta->applicationId, "%x", &nResult);

   ///nach applicationId suchen damit erhält man den dazugehörigen ReqId

       for(int i = 0;i < 4; i++) {
           if(arrayCstaIoRegisterReqId[1][i]== nResult)
           {
               RegisterReqId = arrayCstaIoRegisterReqId[0][i];
               arrayCstaIoRegisterReqId[0][i] = 0;
               arrayCstaIoRegisterReqId[1][i] = 0;
               arrayCstaIoRegisterReqId[2][i] = 0;
               i = 5;
           }
       }
   int i = _oss_encd_uint(0, &data, 0,0, RegisterReqId);
   _oss_encd_uint(0, &data, 0,0, i);
   _oss_encd_uint(0, &data, 0,0, 0x04);
   _oss_encd_uint(0, &data, 0,0, i+2);
   _oss_encd_uint(0, &data, 0,0, 0x30);
   _oss_encd_uint(0, &data, 0,0, 0x56);    //Opcode
   _oss_encd_uint(0, &data, 0,0, 0x01);    //
   _oss_encd_uint(0, &data, 0,0, 0x02);
   _oss_encd_uint(0, &data, 0,0, 0x02);

   _oss_encd_uint(0, &data, 0,0, g_InvokeId);
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x02);

   _oss_encd_uint(0, &data, 0,0, i+11);
   _oss_encd_uint(0, &data, 0,0, 0xA1);

   _oss_encd_uint(0, &data, 0,0, i+13);
   _oss_encd_uint(0, &data, 0,0, 0x00);

   sendTcp(data,i+15);
   generateInvokeId();
}

// a2 0a 02 01 23 30 05 02 01 77 05 00
void PDIAPI sendFastDataResponse(unsigned int invokeId){
   char tmp[1024];
   char pdu[1024];
   char* data=pdu+1024;
   char* base=data;

   int len=scanHex("30 05 02 01 77 05 00",tmp);
   data-=len;
```

```
    memcpy(data,tmp,len);

    // invokeId
    int vintlen=_oss_encd_int(0, &data, 0,0, invokeId);
    _oss_encd_uint(0, &data, 0,0, vintlen);
    _oss_encd_uint(0, &data, 0,0, 0x02);
    // Response
    int roseLen=(int)base-(int)data;
    _oss_encd_uint(0, &data, 0,0, roseLen);
    _oss_encd_uint(0, &data, 0,0, 0xa2);

    //Gesamtlänge
    vintlen=_oss_encd_int(0, &data, 0,0, 2+roseLen);
    _oss_encd_uint(0, &data, 0,0, 0x00);


    //sendLayer4(data,2+roseLen); für Hipath 4000 nicht notwendig
    sendTcp(data,1+vintlen+2+roseLen);
}


// FastDataRequest
//00 2f a1 2d 02 01 32 02 01 77 30 25 30 06 80 04 32 31 32 36 04 03 41 42
43 7e 16 a1 14 bc 12
//(04 0b (53 69 65 6d 65 6e 73 20 43 41 50<-SiemensCAP))
//(b0 03 81 01 01 <- Beep)
void PDIAPI sendFastDataRequest(CstaDataFastRequest* csta)
{
    // Thread aufgerufen wenn diplay fixed ausgewählt ist
    char pdu[1024];
    char* data=pdu+1024;
    char* base=data;
    //int invokeId = 23;
    int audio;

    if (csta->audi==1)
    {
    // enkodieren für beep-signal
    _oss_encd_uint(0,&data,0,0,0x01);
    }
    else _oss_encd_uint(0,&data,0,0,0x00);

    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x81);
    _oss_encd_uint(0,&data,0,0,0x03);
    _oss_encd_uint(0,&data,0,0,0xb0);
    audio = 5;

    // Enkodierung SiemensCAP
    int L1 = _oss_encd_nstr(0,&data,0,0,csta->manufacturer,0);
    _oss_encd_uint(0,&data,0,0,L1);
    _oss_encd_uint(0,&data,0,0,0x04);

    _oss_encd_uint(0,&data,0,0,audio+L1+2);
    _oss_encd_uint(0,&data,0,0,0xbc);
    _oss_encd_uint(0,&data,0,0,audio+L1+4);
    _oss_encd_uint(0,&data,0,0,0xa1);
    _oss_encd_uint(0,&data,0,0,audio+L1+6);
    _oss_encd_uint(0,&data,0,0,0x7e);


    // Ausgabe Daten
```

```
    int L2 = _oss_encd_nstr(0,&data,0,0,csta->ioData,0);
    _oss_encd_uint(0,&data,0,0,L2);
    _oss_encd_uint(0,&data,0,0,0x04);

    //30 25   30 06 80 04 32 31 32 36 -> Device nr
    int L3 = _oss_encd_nstr(0,&data,0,0,csta->device,0);
    _oss_encd_uint(0,&data,0,0,L3);

    _oss_encd_uint(0,&data,0,0,0x80);
    _oss_encd_uint(0,&data,0,0,L3+2);
    _oss_encd_uint(0,&data,0,0,0x30);
    _oss_encd_uint(0,&data,0,0,14+audio+L1+L2+L3);
    _oss_encd_uint(0,&data,0,0,0x30);


    //header //00 2f a1 2d 02 01 32 02 01 77
    _oss_encd_uint(0,&data,0,0,0x77);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,g_InvokeId);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,22+audio+L1+L2+L3);
    _oss_encd_uint(0,&data,0,0,0xA1);
    _oss_encd_uint(0,&data,0,0,24+audio+L1+L2+L3);
    _oss_encd_uint(0,&data,0,0,0x00);
    g_length_fastData = 26+audio+L1+L2+L3;


    sendTcp(data,g_length_fastData);
    generateInvokeId();
}



// Clear Connection
// a1 14 02 01 06 02 01 05 30 0c 6b 0a a1 08 30 06 80 04 32 31 32 36
void PDIAPI sendClearConnectionRequest (CstaClearConnectionRequest *csta)
{
    char pdu[1024];
    char* data=pdu+1024;
    char* base=data;
    //int invokeId = 26;

    int l1 = _oss_encd_nstr(0,&data,0,0,csta->device,0);
    _oss_encd_uint(0,&data,0,0,l1);
    _oss_encd_uint(0,&data,0,0,0x80);

    _oss_encd_uint(0,&data,0,0,l1+2);
    _oss_encd_uint(0,&data,0,0,0x30);

    _oss_encd_uint(0,&data,0,0,l1+4);
    _oss_encd_uint(0,&data,0,0,0xa1);

    _oss_encd_uint(0,&data,0,0,l1+6);
    _oss_encd_uint(0,&data,0,0,0x6B);

    int z1 = l1+8;
    _oss_encd_uint(0,&data,0,0,z1);

    //a1 14 02 01 05 02 01 09 30
    _oss_encd_uint(0,&data,0,0,0x30);
```

```
    _oss_encd_uint(0,&data,0,0,0x05);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,g_InvokeId);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);

     int z2 = z1+8;
    _oss_encd_uint(0,&data,0,0,z2);
    _oss_encd_uint(0,&data,0,0,0xA1);

    _oss_encd_uint(0,&data,0,0,z2+2);
    _oss_encd_uint(0,&data,0,0,0x00);
    sendTcp(data,z2+4);
    generateInvokeId();
}


//Hold Call
//a1 14 02 01 05 02 01 09 30 0c 6b 0a a1 08 30 06 80 04 32 31 32 36 <--
callId
void PDIAPI sendHoldCallRequest(CstaHoldCallRequest *csta)
{
    char pdu[1024];
    char* data=pdu+1024;
    char* base=data;
    //int invokeId = 24;

    int l1 = _oss_encd_nstr(0,&data,0,0,csta->device,0);
    _oss_encd_uint(0,&data,0,0,l1);
    _oss_encd_uint(0,&data,0,0,0x80);

    _oss_encd_uint(0,&data,0,0,l1+2);
    _oss_encd_uint(0,&data,0,0,0x30);

    _oss_encd_uint(0,&data,0,0,l1+4);
    _oss_encd_uint(0,&data,0,0,0xa1);

    _oss_encd_uint(0,&data,0,0,l1+6);
    _oss_encd_uint(0,&data,0,0,0x6b);

    int z1 = l1+8;
    _oss_encd_uint(0,&data,0,0,z1);


    //a1 14 02 01 05 02 01 09 30
    _oss_encd_uint(0,&data,0,0,0x30);
    _oss_encd_uint(0,&data,0,0,0x09);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,g_InvokeId);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);

     int z2 = z1+8;
    _oss_encd_uint(0,&data,0,0,z2);
    _oss_encd_uint(0,&data,0,0,0xA1);

    _oss_encd_uint(0,&data,0,0,z2+2);
    _oss_encd_uint(0,&data,0,0,0x00);
    sendTcp(data,z2+4);
    generateInvokeId();
```

```
}



// Conference Call
// a1 20 02 01 04 02 01 06 30 18 6b 0a a1 08 30 06 80 04 32 31 32 36 <--
hold call
// 6b 0a a1 08 30 06 80 04 32 32 30 31 <- active Call
void PDIAPI sendConferenceCallRequest(CstaConferenceCallRequest *csta)
{

   char pdu[1024];
   char* data=pdu+1024;
   char* base=data;
   //int invokeId = 22;

   int l1 = _oss_encd_nstr(0,&data,0,0,csta->activeCallId,0);
   _oss_encd_uint(0,&data,0,0,l1);
   _oss_encd_uint(0,&data,0,0,0x80);

   _oss_encd_uint(0,&data,0,0,l1+2);
   _oss_encd_uint(0,&data,0,0,0x30);

   _oss_encd_uint(0,&data,0,0,l1+4);
   _oss_encd_uint(0,&data,0,0,0xa1);

   _oss_encd_uint(0,&data,0,0,l1+6);
   _oss_encd_uint(0,&data,0,0,0x6B);

   int l2 = _oss_encd_nstr(0,&data,0,0,csta->heldCallId,0);
   _oss_encd_uint(0,&data,0,0,l2);
   _oss_encd_uint(0,&data,0,0,0x80);

   _oss_encd_uint(0,&data,0,0,l2+2);
   _oss_encd_uint(0,&data,0,0,0x30);

   _oss_encd_uint(0,&data,0,0,l2+4);
   _oss_encd_uint(0,&data,0,0,0xa1);

   _oss_encd_uint(0,&data,0,0,l2+6);
   _oss_encd_uint(0,&data,0,0,0x6B);

   int z1 = l1+l2+16;
   _oss_encd_uint(0,&data,0,0,z1);

   _oss_encd_uint(0,&data,0,0,0x30);
   _oss_encd_uint(0,&data,0,0,0x06);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);
   _oss_encd_uint(0,&data,0,0,g_InvokeId);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);

   _oss_encd_uint(0,&data,0,0,z1+7);
   _oss_encd_uint(0,&data,0,0,0xA1);

   _oss_encd_uint(0,&data,0,0,z1+9);
```

```
   _oss_encd_uint(0,&data,0,0,0x00);

   sendTcp(data,z1+11);
   generateInvokeId();
}

//Park Call
//a1 15 02 01 04 02 01 12 30 0D 6B 03 80 01 05 (30 06 80 04 32 31 32 36)
void sendParkCallRequest (CstaParkCallRequest *csta)
{
   char pdu[1024];
   char* data=pdu+1024;
   char* base=data;
   //int invokeId = 26;


   int l1 = _oss_encd_nstr(0,&data,0,0,csta->dailingdevice,0);
   _oss_encd_uint(0,&data,0,0,l1);
   _oss_encd_uint(0,&data,0,0,0x80);

   _oss_encd_uint(0,&data,0,0,l1+2);
   _oss_encd_uint(0,&data,0,0,0x30);

   _oss_encd_uint(0,&data,0,0,l1+4);
   _oss_encd_uint(0,&data,0,0,0xa1);

   _oss_encd_uint(0,&data,0,0,l1+6);
   _oss_encd_uint(0,&data,0,0,0x6B);

   int l2 = _oss_encd_nstr(0,&data,0,0,csta->callId,0);
   _oss_encd_uint(0,&data,0,0,l2);
   _oss_encd_uint(0,&data,0,0,0x80);

   _oss_encd_uint(0,&data,0,0,0x03);
   _oss_encd_uint(0,&data,0,0,0x6B);

   int z1=l1+l2+12;
   _oss_encd_uint(0,&data,0,0,z1);

   _oss_encd_uint(0,&data,0,0,0x12);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);
   _oss_encd_uint(0,&data,0,0,g_InvokeId);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);

   _oss_encd_uint(0,&data,0,0,z1+19);
   _oss_encd_uint(0,&data,0,0,0xA1);

   _oss_encd_uint(0,&data,0,0,z1+21);
   _oss_encd_uint(0,&data,0,0,0x00);


   sendTcp(data,z1+2+23);
   generateInvokeId();
}
//Alternate Call
//a1 20 02 01 03 02 01 01 30 18 6b 0a a1 08 30 06 80 04 (32 32 31 30) <-
held connection
//6b 0a a1 08 30 06 80 04 (32 31 32 36)<--active connection
void PDIAPI sendAlternateCallRequest(CstaAlternateCallRequest *csta)
{
```

```
    char pdu[1024];
    char* data=pdu+1024;
    char* base=data;
    //int invokeId = 20;


    int l1 = _oss_encd_nstr(0,&data,0,0,csta->activeCallId,0);
    _oss_encd_uint(0,&data,0,0,l1);
    _oss_encd_uint(0,&data,0,0,0x80);

    _oss_encd_uint(0,&data,0,0,l1+2);
    _oss_encd_uint(0,&data,0,0,0x30);

    _oss_encd_uint(0,&data,0,0,l1+4);
    _oss_encd_uint(0,&data,0,0,0xa1);

    _oss_encd_uint(0,&data,0,0,l1+6);
    _oss_encd_uint(0,&data,0,0,0x6B);

    int l2 = _oss_encd_nstr(0,&data,0,0,csta->heldCallId,0);
    _oss_encd_uint(0,&data,0,0,l2);
    _oss_encd_uint(0,&data,0,0,0x80);

    _oss_encd_uint(0,&data,0,0,l2+2);
    _oss_encd_uint(0,&data,0,0,0x30);

    _oss_encd_uint(0,&data,0,0,l2+4);
    _oss_encd_uint(0,&data,0,0,0xa1);

    _oss_encd_uint(0,&data,0,0,l2+6);
    _oss_encd_uint(0,&data,0,0,0x6B);

    int z1 = l1+l2+16;
    _oss_encd_uint(0,&data,0,0,z1);

    //a1 20 02 01 03 02 01 01 30
    _oss_encd_uint(0,&data,0,0,0x30);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,g_InvokeId);

    _oss_encd_uint(0,&data,0,0,z1+5);
    _oss_encd_uint(0,&data,0,0,0xA1);

    _oss_encd_uint(0,&data,0,0,z1+7);
    _oss_encd_uint(0,&data,0,0,0x00);

    sendTcp(data,z1+9);
    generateInvokeId();
}


//SetDisplayRequest
//00 18 (A1 16 02 01 07 02 02 01 12 30 0d 30 06 80 04 (32 31 32 36)<-Callid
16 03 (41 42 43<-Text))

void PDIAPI sendSetDisplayRequest(CstaSetDisplayRequest* csta)
{
    //char tmp[1024];
    char pdu[1024];
    char* data=pdu+1024;
```

```
    char* base=data;

    //int invokeId = 12;

    //Text encodieren + Rufnummer
    int l1 = _oss_encd_nstr(0,&data,0,0,csta->display,0);
    _oss_encd_uint(0,&data,0,0,l1);
    _oss_encd_uint(0,&data,0,0,0x16);
    int l2 = _oss_encd_nstr(0,&data,0,0,csta->device,0);
    _oss_encd_uint(0,&data,0,0,l2);
    _oss_encd_uint(0,&data,0,0,0x80);
    _oss_encd_uint(0,&data,0,0,l2+2);
    _oss_encd_uint(0,&data,0,0,0x30);
    _oss_encd_uint(0,&data,0,0,l1+l2+6);
    _oss_encd_uint(0,&data,0,0,0x30);

// Rose-Header kodieren
    _oss_encd_uint(0,&data,0,0,0x12);
    _oss_encd_uint(0,&data,0,0,0x01);// <- Opcode 0x112
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,0x02);

    _oss_encd_uint(0,&data,0,0,g_InvokeId);
    _oss_encd_uint(0,&data,0,0,0x01);
    _oss_encd_uint(0,&data,0,0,0x02);
    _oss_encd_uint(0,&data,0,0,l1+l2+15);
    _oss_encd_uint(0,&data,0,0,0xA1);
    _oss_encd_uint(0,&data,0,0,l1+l2+17);
    _oss_encd_uint(0,&data,0,0,0x00);


    sendTcp(data,l1+l2+19);
    generateInvokeId();

}
// Monitor Start Request mit CC Filter
//a1 1b 02 01 1b 02 01 47 30 13 30 06 80 04 (32 31 32 36)<- Device number
//a0 06 80 04 (06 80 01 80)<- Filter 0a 01 01  <- M.F.
void PDIAPI sendMonitorStartRequest(CstaMonitorStartRequest* csta) {
    //char filter[1024];
    char pdu[1024];
    char* data=pdu+1024;
    int length;
    //int invokeId = 30;
    int filter1 = 0x00;
    int filter2 = 0x00;
    //char cc_filter[4];
    //char* pcc_filter = cc_filter+4;


    char* temp;

    temp = csta->device;



    int index = atoi(temp);
    g_index_device = index;

    _oss_encd_uint(0, &data, 0,0, 0x01);
    _oss_encd_uint(0, &data, 0,0, 0x01);
    _oss_encd_uint(0, &data, 0,0, 0x0a);
```

```
    // Physical Device Events
    if (csta->physicalButtonPress == 1)  filter1 = filter1 + 0x40;
    if (csta->physicalButtonInfo == 1)  filter1 = filter1 + 0x80;
    if (csta->physicalDisplayUpdated == 1)  filter1 = filter1 + 0x20;
    if (csta->physicalHookSwitch == 1)  filter1 = filter1 + 0x10;
    if (csta->physicalLampMode == 1)  filter1 = filter1 + 0x08;

    _oss_encd_uint(0, &data, 0,0, filter1);
    _oss_encd_uint(0, &data, 0,0, 0x00);
    _oss_encd_uint(0, &data, 0,0, 0x02);
    _oss_encd_uint(0, &data, 0,0,  0x88);
    filter1 = 0x00;
    // Call Controll Device Events
    if (csta->callBridged == 1)  filter1 = filter1 + 0x01;
    if (csta->callCleard == 1)  filter2 = filter2 + 0x80;
    if (csta->callConnectionCleared == 1)  filter2 = filter2 + 0x20;
    if (csta->callDelivered == 1)  filter2 = filter2 + 0x10;
    if (csta->callEstablished == 1)  filter2 = filter2 + 0x04;
    if (csta->callDiverted == 1)  filter2 = filter2 + 0x08;
    if (csta->callFailed == 1)  filter2 = filter2 + 0x02;
    if (csta->callHeld == 1)  filter1 = filter1 + 0x01;
    if (csta->callQueued == 1)  filter1 = filter1 + 0x20;
    if (csta->callRetrieved == 1)  filter1 = filter1 + 0x10;
    if (csta->callDigitsDailed == 1)  filter1 = filter1 + 0x02;
    if (csta->CallConferenced == 1 ) filter2 = filter2 + 0x40;
    _oss_encd_uint(0, &data,0,0,filter1);
    _oss_encd_uint(0, &data,0,0,filter2);
    filter1 = 0x00;
    filter2 = 0x00;
    _oss_encd_uint(0, &data,0,0,0x00);
    _oss_encd_uint(0, &data, 0,0, 0x03);
    _oss_encd_uint(0, &data, 0,0, 0x80);
    _oss_encd_uint(0, &data, 0,0,0x09);
    _oss_encd_uint(0, &data, 0,0, 0xa0);


    int l1 =_oss_encd_nstr(0, &data, 0,0, csta->device, 0);
    _oss_encd_uint(0, &data, 0,0, l1);
    _oss_encd_uint(0, &data, 0,0, 0x80);
    _oss_encd_uint(0, &data, 0,0, l1 + 2);

    //a1 1b 02 01 1b 02 01 47 30 13 30

    _oss_encd_uint(0, &data, 0, 0, 0x30);
    _oss_encd_uint(0, &data, 0, 0, 18+l1);
    _oss_encd_uint(0, &data, 0, 0, 0x30);
    _oss_encd_uint(0, &data, 0, 0, 0x47);
    _oss_encd_uint(0, &data, 0, 0, 0x01);
    _oss_encd_uint(0, &data, 0, 0, 0x02);
    _oss_encd_uint(0, &data, 0, 0, g_InvokeId);
    _oss_encd_uint(0, &data, 0, 0, 0x01);
    _oss_encd_uint(0, &data, 0, 0, 0x02);
    _oss_encd_uint(0, &data, 0, 0, 26+l1);
    _oss_encd_uint(0, &data, 0, 0, 0xA1);
    _oss_encd_uint(0, &data, 0, 0, 28+l1);
    _oss_encd_uint(0, &data, 0, 0, 0x00);

    sendTcp(data,30+l1);
    generateInvokeId();
}
```

```
void PDIAPI sendMonitorStopRequest(CstaMonitorStopRequest *csta){
   char pdu[1024];
   char* data=pdu+1024;
   //int suchDevice;
   //int refId;
   //int invokeId=30;



   for (int i= 0 ; i < zeileMonitorRefId+1; i++)
   {
       if (g_index_device == arrayCstaMonitorRefId[0][i])
       {
           csta->refIdMonitor=arrayCstaMonitorRefId[1][i];
           g_lineDelete = i;
           i = zeileMonitorRefId+1;
       }

   }
   //a1 0b 02 01 0e 02 01 49 30 03 55 01 07
   if (csta->refIdMonitor>0)
   {

   _oss_encd_uint(0,&data,0,0,csta->refIdMonitor);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x55);
   _oss_encd_uint(0,&data,0,0,3);
   _oss_encd_uint(0,&data,0,0,0x30);

   _oss_encd_uint(0,&data,0,0,0x49);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);
   _oss_encd_uint(0,&data,0,0,g_InvokeId);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);
   _oss_encd_uint(0,&data,0,0,11);
   _oss_encd_uint(0,&data,0,0,0xA1);
   _oss_encd_uint(0,&data,0,0,13);
   _oss_encd_uint(0,&data,0,0,0x00);
   sendTcp(data,15);
   generateInvokeId();

   arrayCstaMonitorRefId[0][g_lineDelete] = 0;
   arrayCstaMonitorRefId[1][g_lineDelete] = 0;

   }
}


// Snapshot Device
//a1 10 02 01 09 02 01 4a 30 08   30 06 80 04 32 31 32 36
void PDIAPI sendSnapshotDeviceRequest (CstaSnapshotDeviceRequest *csta)
{
   char pdu[1024];
   char* data=pdu+1024;
   int opcode = 0x4A;

   //int invokeId=40;

   int l1 = _oss_encd_nstr(0,&data,0,0,csta->deviceId,0);
   _oss_encd_uint(0,&data,0,0,l1);
   _oss_encd_uint(0,&data,0,0,0x80);
```

```
   _oss_encd_uint(0,&data,0,0,l1+2);
   _oss_encd_uint(0,&data,0,0,0x30);
   _oss_encd_uint(0,&data,0,0,l1+4);
   _oss_encd_uint(0,&data,0,0,0x30);
   _oss_encd_uint(0,&data,0,0,opcode); //<- opcode
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);
   _oss_encd_uint(0,&data,0,0,g_InvokeId);
   _oss_encd_uint(0,&data,0,0,0x01);
   _oss_encd_uint(0,&data,0,0,0x02);
   _oss_encd_uint(0,&data,0,0,l1+12);
   _oss_encd_uint(0,&data,0,0,0xA1);
   _oss_encd_uint(0,&data,0,0,l1+14);
   _oss_encd_uint(0,&data,0,0,0x00);

   sendTcp(data,l1+16);
   generateInvokeId();
}

// A1 13 02 01 02 02 02 01 04 30 0A 30 05 80 03 31 30 30 04 01 15
void PDIAPI sendButtonPressRequest(char* device,int button) {
   char pdu[1024];
   char* data=pdu+1024;
   _oss_encd_uint(0, &data, 0,0, button);
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x04);
   // device
   _oss_encd_nstr(0, &data, 0,0, device, 0);
   _oss_encd_uint(0, &data, 0,0, strlen(device));
   _oss_encd_uint(0, &data, 0,0, 0x80);
   _oss_encd_uint(0, &data, 0,0, 2+strlen(device));
   _oss_encd_uint(0, &data, 0,0, 0x30);
   _oss_encd_uint(0, &data, 0,0, 2+2+strlen(device)+3);
   _oss_encd_uint(0, &data, 0,0, 0x30);
   sendRoseRequest(data,2+2+2+strlen(device)+3,0x104,0x104);
}

//30 21 30 05 80 03 31 30 30 16 18 48 61 6C 6C 6F 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20
/*
void sendSetDisplayRequest(char* device,char* display) {
   char pdu[1024];
   char* data=pdu+1024;
   _oss_encd_nstr(0, &data, 0,0, display, 0);
   _oss_encd_uint(0, &data, 0,0, strlen(display));
   _oss_encd_uint(0, &data, 0,0, 0x16);
   _oss_encd_nstr(0, &data, 0,0, device, 0);
   _oss_encd_uint(0, &data, 0,0, strlen(device));
   _oss_encd_uint(0, &data, 0,0, 0x80);
   _oss_encd_uint(0, &data, 0,0, strlen(device)+2);
   _oss_encd_uint(0, &data, 0,0, 0x30);
   _oss_encd_uint(0, &data, 0,0, strlen(device)+2+2+strlen(display)+2);
   _oss_encd_uint(0, &data, 0,0, 0x30);

   sendRoseRequest(data,2+strlen(device)+2+2+strlen(display)+2,0x0112);
}
*/
// M.F 30 07 A0 03 04 01 01 04 <- typ XX <-länge xx<- String

/*
a1 2e 02 01 04 02 01 70 30 26 a0 03 04 01 02 04 04 74 65 73
74 7e 19 a1 17 bc 15 04 0b 53 69 65 6d 65 6e 73 20 43 41 50 b0 06
```

```
80 01 01<- fixed 81 01 01 <- beep(not silent)
*/
void PDIAPI sendSendDataRequest(CstaSendDataRequest* csta) {
   char pdu[1024];
   char* data=pdu+1024;
   int z1 = 0;
   int z2 = 0;

       if(csta->silent == 0)
           {
           _oss_encd_uint(0, &data, 0,0, 0x01);
           }
       else _oss_encd_uint(0, &data, 0,0 ,0x00);

   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x81);
   z1 = 3;


       if(csta->fixed == 1)
           {
           _oss_encd_uint(0, &data, 0,0, 0x01);
           }

       else _oss_encd_uint(0, &data, 0,0, 0x00);

   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x80);
   z2 = 3;

   _oss_encd_uint(0, &data, 0,0, z1+z2);
   _oss_encd_uint(0, &data, 0,0, 0xB0);

   //04 0b 53 69 65 6d 65 6e 73 20 43 41 50 <- Siemens CAP

   int lenManu = _oss_encd_nstr(0, &data, 0,0, csta->manufacturer,0);
   _oss_encd_uint(0, &data, 0,0, lenManu);
   _oss_encd_uint(0, &data, 0,0, 0x04);

   //7e 19 a1 17 bc 15
   _oss_encd_uint(0, &data, 0,0, z1+z2+lenManu+4);
   _oss_encd_uint(0, &data, 0,0, 0xBC);
   _oss_encd_uint(0, &data, 0,0, z1+z2+lenManu+6);
   _oss_encd_uint(0, &data, 0,0, 0xA1);
   _oss_encd_uint(0, &data, 0,0, z1+z2+lenManu+8);
   _oss_encd_uint(0, &data, 0,0, 0x7E);

   //04 04 74 65 73 74
   int lenData = _oss_encd_nstr(0, &data, 0,0, csta->ioData,0);
   _oss_encd_uint(0, &data, 0,0, lenData);
   _oss_encd_uint(0, &data, 0,0, 0x04);

   //a0 03 04 01 02
   _oss_encd_uint(0, &data, 0,0, csta->crossRefId);
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x04);
   _oss_encd_uint(0, &data, 0,0, 0x03);
   _oss_encd_uint(0, &data, 0,0, 0xA0);


   //30 26
   _oss_encd_uint(0, &data, 0,0, z1+z2+lenManu+lenData+17);
```

```
   _oss_encd_uint(0, &data, 0,0, 0x30);

   //02 01 70
   _oss_encd_uint(0, &data, 0,0, 0x70); //<- opcode
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x02);

   //02 01 04
   _oss_encd_uint(0, &data, 0,0, g_InvokeId); //<- invokeId
   _oss_encd_uint(0, &data, 0,0, 0x01);
   _oss_encd_uint(0, &data, 0,0, 0x02);

   //00 30 a1 2e
   _oss_encd_uint(0, &data, 0,0,z1+z2+lenManu+lenData+25);
   _oss_encd_uint(0, &data, 0,0, 0xA1);
   _oss_encd_uint(0, &data, 0,0, z1+z2+lenManu+lenData+27);
   _oss_encd_uint(0, &data, 0,0, 0x00);

   sendTcp(data,z1+z2+lenManu+lenData+29);
   generateInvokeId();

}



// A1 34 02 01 08 02 01 70 30 2C A0 03 04 01 06 04 10 54 45 53 54 54 45 53
54 54 45 53 54 54 45 53 54 7E 13 A1 11 E1 0F 06 06 2B 0C 02 88 53 0F 04 05
E0 03 F5 01 30
// 26 80 34 A1 32 02 01 63 02 01 70 30 28 A0 03 04 01 64 04 0E 23 20 50 49
4E 2D 45 69 6E 67 61 62 65 3A            7E 13 A1 11 E1 0F 06 06 2B 0C 02
88 53 0F 04 05 E0 03 F5 01 30
/*
void sendSendDataRequest(CstaSendDataRequest* csta) {
// char tmp[1024];
   char pdu[1024];
   char* data=pdu+1024;
   char* base=data;
   char* IoData = csta->ioData;
   char* Device = csta->device;

   // Next Parameter, Accept and Echo Input, SubApp=48
   //int extlen=scanHex("7E 13 A1 11 E1 0F 06 06 2B 0C 02 88 53 0F 04 05 E0
03 F5 01 30",tmp);
   // Next Parameter, Accept and Echo Asteriks, SubApp=48
/* int extlen=scanHex("7E 13 A1 11 E1 0F 06 06 2B 0C 02 88 53 0F 04 05 E0
03 F5 03 30",tmp);
   data-=extlen;
   memcpy(data,tmp,extlen);
   // data
   _oss_encd_nstr(0, &data, 0,0, display, 0);
   _oss_encd_uint(0, &data, 0,0, strlen(display));
   _oss_encd_uint(0, &data, 0,0, 0x04);
   // crossRefId
   int vlen=_oss_encd_int(0, &data, 0,0, g_ioCrossRefId);
   _oss_encd_uint(0, &data, 0,0, vlen);
   _oss_encd_uint(0, &data, 0,0, 0x04);
   _oss_encd_uint(0, &data, 0,0, 2+vlen);
   _oss_encd_uint(0, &data, 0,0, 0xa0);
   // 0x30
   _oss_encd_uint(0, &data, 0,0, 2+2+vlen+2+strlen(display)+extlen);
   _oss_encd_uint(0, &data, 0,0, 0x30);
```

```
   sendRoseRequest(data,2+2+2+vlen+2+strlen(display)+extlen,0x70,0x63);
*/ //<- M.F. 3.2

   //30 08 A0 03 04 01 01 04<- typ XX <-länge xx<- String
   //30 0B A0 03 04 01 09 04 04 34 35 34 35
/*
   char applicationId[] = "559";
   char manufacturer[] = "Siemens CAP";
   // Private data
   _oss_encd_nstr(0, &data, 0,0, applicationId, 0);
   _oss_encd_uint(0, &data, 0,0, strlen(applicationId));
   _oss_encd_uint(0, &data, 0,0, 0x82);
   _oss_encd_uint(0, &data, 0,0, 2+strlen(applicationId));
   _oss_encd_uint(0, &data, 0,0, 0xb0);
   _oss_encd_nstr(0, &data, 0,0, manufacturer, 0);
   _oss_encd_uint(0, &data, 0,0, strlen(manufacturer));
   _oss_encd_uint(0, &data, 0,0, 0x04);
   _oss_encd_uint(0, &data, 0,0,
strlen(applicationId)+2+2+strlen(manufacturer)+2);
   _oss_encd_uint(0, &data, 0,0, 0xbc);
   _oss_encd_uint(0, &data, 0,0,
2+strlen(applicationId)+2+2+strlen(manufacturer)+2);
   _oss_encd_uint(0, &data, 0,0, 0xa1);
   _oss_encd_uint(0, &data, 0,0,
2+2+strlen(applicationId)+2+2+strlen(manufacturer)+2);
   _oss_encd_uint(0, &data, 0,0, 0x7e);
   // Io DATA
   _oss_encd_nstr(0,&data, 0, 0, IoData, 0);
   int l1 = strlen(IoData);
   _oss_encd_uint(0,&data, 0, 0, l1);
   _oss_encd_uint(0,&data, 0, 0, 0x04);
   _oss_encd_uint(0,&data, 0, 0, g_ioCrossRefId);
   _oss_encd_uint(0,&data, 0, 0, 0x01);
   _oss_encd_uint(0,&data, 0, 0, 0x04);
   _oss_encd_uint(0,&data, 0, 0, 0x03);
   _oss_encd_uint(0,&data, 0, 0, 0xA0);
   _oss_encd_uint(0,&data, 0, 0, 7+l1 +
2+2+strlen(applicationId)+2+2+strlen(manufacturer)+2);//<-Länge
   _oss_encd_uint(0,&data, 0, 0, 0x30);

   _oss_encd_uint(0,&data, 0, 0, 1+7+l1 +
2+2+strlen(applicationId)+2+2+strlen(manufacturer)+2);
   _oss_encd_uint(0,&data, 0, 0,0xA1);

   _oss_encd_uint(0,&data, 0, 0, 1+2+9+l1 +
2+2+strlen(applicationId)+2+2+strlen(manufacturer)+2);
   _oss_encd_uint(0,&data, 0, 0, 0x00);
   sendTcp(data,2+2+11+l1 +
2+2+strlen(applicationId)+2+2+strlen(manufacturer)+2);
}
*/

void PDIAPI sendCstaRequest(CstaRequest* message) {
   switch (message->opValue) {
   case 0x000a:
       sendMakeCallRequest(&(message->argument.makeCallRequest));
   }
}


//30 09 A1 03 04 01 04 04 02 31 38
void recvSendDataRequest(char* data,int len,CstaRequest* csta) {
```

```
    char* base=data;
    unsigned char tag;
    unsigned char parlen;
    unsigned char taglen;
    strcpy(csta->argument.sendDataRequest.ioData,"");
    // 30 09
    tag=*(data++);
    parlen=*(data++);
    // A1 33
    tag=*(data++);
    taglen=*(data++);
    // 04 01
    tag=*(data++);
    taglen=*(data++);
    csta->argument.sendDataRequest.crossRefId
=_oss_dec_iint(0,&data,0,taglen);
    if (parlen>(int)(data-base-2)) {
        tag=*(data++);
        taglen=*(data++);
        _oss_dec_nstr(0, &data,0,taglen, csta-
>argument.sendDataRequest.ioData, 10,0);
    }
    recvCstaRequest(csta);
}

//30 1C 30 05 80 03 31 30 30 7E 13 A1 11 E1 0F 06 06 2B 0C 02 88 53 0F 04
05 E0 03 FF FF 31
void recvStartDataPathRequest(char* data,int len,CstaStartDataPathRequest*
csta) {
    unsigned char tag;
    unsigned char taglen;
    //
    tag=*(data++);
    taglen=*(data++);
    tag=*(data++);
    taglen=*(data++);
    tag=*(data++);
    taglen=*(data++);
    _oss_dec_nstr(0, &data,0,taglen, csta->device, 10,0);
}

//30 05 a0 03 04 01 01
void recvStartDataPathResponse(char* data, int len, CstaStartDataPathRe-
sponse* csta, int invokeId){
    unsigned char tag;
    unsigned char taglen;
    //
    tag=*(data++);
    taglen=*(data++);
    tag=*(data++);
    taglen=*(data++);
    tag=*(data++);
    taglen=*(data++);

    csta->ioCrossRefId = _oss_dec_iint(0,&data,0,taglen);
    /*
    int arrayCstaIoCrossRefId[3][30];
    int zeileIoCrossRefId = 0;
    */
                for (int i=0;i<29;i++) {
                    if(arrayCstaIoCrossRefId[2][i] == invokeId)
                    {
```

```
                        arrayCstaIoCrossRefId[0][i] = csta->ioCrossRefId;
                        i = 30;
                    }
            }

    //g_ioCrossRefId= csta->ioCrossRefId;
}


//30 05 A1 03 04 01 01
void recvStopDataPathRequest(char* data,int len,CstaStopDataPathRequest*
csta) {
    unsigned char tag;
    unsigned char taglen;

    tag=*(data++);
    taglen=*(data++);
    tag=*(data++);
    taglen=*(data++);
    tag=*(data++);
    taglen=*(data++);
    csta->ioCrossRefId =_oss_dec_iint(0,&data,0,taglen);
}

//30 3A 55 02 00 00 A3 34 A1 32 63 07 30 05 80 03 31 30 30 04 01 F2
void recvButtonPressEvent(char* data,int len,CstaButtonPressEvent* csta) {
    unsigned char tag;
    unsigned char taglen;
    /*
    tag=*(data++);
    taglen=*(data++);
    tag=*(data++);
    taglen=*(data++);
    unsigned int crossRefId =_oss_dec_iint(0,&data,0,taglen);
    // A3 34 //PhysicalDeviceFeatureEvents
    tag=*(data++);
    taglen=*(data++);
    // A1 32
    tag=*(data++);
    taglen=*(data++);*/
    // 63 07
    tag=*(data++);
    taglen=*(data++);
    // 30 05
    tag=*(data++);
    taglen=*(data++);
    // 80 03
    tag=*(data++);
    taglen=*(data++);
    _oss_dec_nstr(0, &data,0,taglen, csta->device, 10,0);
    tag=*(data++);
    taglen=*(data++);
    csta->button=_oss_dec_iint(0,&data,0,taglen);
}

// Verteiler für Response-Dekodierungsfunktionen
void recvRoseResponse(char* data,int len) {
    CstaRequest csta;
    unsigned char tag;
    unsigned char taglen;
    int invokeId;
    tag=*(data++);
```

```
   taglen=*(data++);
   csta.invokeId=_oss_dec_iint(0,&data,0,taglen);
   invokeId = csta.invokeId;
   tag=*(data++);
   taglen=*(data++);
   if (tag==0x30) {
       tag=*(data++);
       taglen=*(data++);
       //return;
   }
   csta.opValue=_oss_dec_iint(0,&data,0,taglen);
   switch (csta.opValue) {
/* case 0xd3: //SystemStatus
       recvCstaRequest(&csta);
       debug.Format("SystemStatus");
       break;*/
   case 340: //IORegister
       recvCstaIoRegisterRe-
sponse(data,len,&csta.argument.ioRegisterResponse,invokeId);
       //debug.Format("IoRegisterResponse");
       break;
   case 110:
       recvStartDataPathResponse( data, len,
&csta.argument.startDataPathResponse,invokeId);
       break;
   case 71:
       recvMonitorStartResponse(data,len,
&csta.argument.monitorStartResponse);
       break;
   case 74:
       recvSnapshotDeviceRe-
sponse(data,len,&csta.argument.snapshotDeviceResponse);
       break;
   default:;
       recvCstaResponse(&csta);
   }
}

//a2 0e 02 01 01 30 09 02 02 01 54 30 03 04 01 01 -- Hipath 4000
void recvCstaIoRegisterResponse(char* data,int len,CstaIoRegisterResponse*
csta, int invokeId){
   unsigned char tag;
   unsigned char taglen;
   tag=*(data++);
   taglen=*(data++);
   if(tag==0x30){
       tag=*(data++);
       taglen=*(data++);
       if(tag==0x04){
           csta->ioRegisterReqId=_oss_dec_iint(0,&data,0,taglen);
           // in Tabelle ioRegisterReqId ergänzen bei gleicher InvokeId

           for (int i=0;i<4;i++) {
                   if(arrayCstaIoRegisterReqId[2][i] == invokeId)
                   {

arrayCstaIoRegisterReqId[0][i] = csta->ioRegisterReqId;
                       i = 6;
                   }

           }
```

```
            //-----Ausgabe von RegisterId---------
            char *s = (char *)malloc(1000);
            sprintf(s,"ReqId=%d", csta->ioRegisterReqId);
            AfxMessageBox(s);
            //-----------------------------------
        }
    }
}

//MonitorStart Request wird ausgewertet
//a2 2f 02 01 1c 30 2a 02 01 47   30 25 55 01 03<- refIdMonitor a0 20 80 04
06 f8 01 c0 86 02 03 48 87 02 06   c0 88 03 05 fb e0 89 03 02 03 60 83 02
05 20 85 02 01 fe
void recvMonitorStartResponse(char* data, int len, CstaMonitorStartRe-
sponse* csta){
unsigned char tag;
unsigned char taglen;
tag=*(data++);
taglen=*(data++);



    if(tag==0x30)
    {
    tag=*(data++);
    taglen=*(data++);
    //tag=*(data++);
    csta->refIdMonitor =_oss_dec_iint(0,&data,0,taglen);


        if(zeileMonitorRefId>=20) zeileMonitorRefId=0;

    arrayCstaMonitorRefId[0][zeileMonitorRefId] = g_index_device;;
    arrayCstaMonitorRefId[1][zeileMonitorRefId] = csta->refIdMonitor;
    zeileMonitorRefId++;

    }
}

// Snapshot Device Response
//a2 0d 02 01 0a 30 08 02 01 4a   30 03 04 01 32
// a2 0d 02 01 09 30 08 02 01 4a   30 03 04 01 31
void recvSnapshotDeviceResponse(char* data, int len, CstaSnapshotDeviceRe-
sponse* csta){
    unsigned char tag;
    unsigned char taglen;

    tag=*(data++);

    if(tag==0x30){
    taglen=*(data++);
    tag=*(data++);
    taglen= *(data++);

    _oss_dec_nstr(0, &data,0,taglen,csta->Service_Cross_referenceID, 10,0);
    }

}
/*
a1 39 02 02 00 bd
02 01 4d 30
30 erste tag ist dann hier
```

```
04 02 31 30 <- Service Cross referenceID
02 01 01 01 01 ff
76 24 30 22 6b 0a a1 08 30   06
80 04 (32 31 32 36) DeviceId
a0 03 4e 01 00 a0 0f 03 01 00 03 01 00   03 01 00 03 01 00 03 01 00 */
void recvCstaSnapshotDeviceDataRequest(char* data, int len, CstaSnapshotDe-
viceDataRequest* csta){
   unsigned char tag;
   unsigned char taglen;
   unsigned char value;
   //30 30 04
   tag = *(data++);
   taglen = *(data++);
   tag = *(data++);

   //02 31 30
   taglen = *(data++);
   _oss_dec_nstr(0, &data,0,taglen,csta->ServiceCrossRefId, 10,0);

   //02 01 01
   tag = *(data++);
   taglen = *(data++);
   value = *(data++);

   //01 01 FF
   tag = *(data++);
   taglen = *(data++);
   value = *(data++);

   //76 24
   tag = *(data++);
   taglen= *(data++);

   //30 22
   tag = *(data++);
   taglen = *(data++);

   //6b 0a
   tag = *(data++);
   taglen = *(data++);

   //a1 08 30 06

   // conid 30 0d 80 01 26
   tag = *(data++);

   if (tag == 0x30)
       {
       taglen = *(data++);
       tag = *(data++);
       taglen = *(data++);
       csta->connId=_oss_dec_iint(0,&data,0,taglen);
       tag = *(data++);
       }

   taglen = *(data++);
   tag = *(data++);
   taglen = *(data++);

   //80 04 (32 31 32 36) DeviceId
   tag = *(data++);
   taglen = *(data++);
```

```
_oss_dec_nstr(0, &data,0,taglen,csta->callId, 10,0);


//a0 03 4e 01 00
tag = *(data++);
taglen = *(data++);
tag = *(data++);
taglen = *(data++);

int temp = _oss_dec_iint(0,&data,0,taglen);

switch(temp){

    case 0:
        csta->callState = "Null";
        break;
    case 1:
        csta->callState = "Pending";
        break;
    case 3:
        csta->callState = "Originated";
        break;
    case 35:
        csta->callState = "Delivered";
        break;
    case 36:
        csta->callState = "DeliveredHeld";
        break;
    case 50:
        csta->callState = "Received";
        break;
    case 51:
        csta->callState = "Established";
        break;
    case 52:
        csta->callState = "EstablishedHeld";
        break;
    case 66:
        csta->callState = "ReceivedOnHold";
        break;
    case 67:
        csta->callState = "EstablishedOnHold";
        break;
    case 83:
        csta->callState = "Queued";
        break;
    case 84:
        csta->callState = "QueuedHeld";
        break;
    case 99:
        csta->callState = "Failed";
        break;
    case 100:
        csta->callState = "FailedHeld";
        break;
    case 96:
        csta->callState = "Blocked";
        break;

    default:
        break;
}
```

```
}

//a1 35 02 01 02 02 01 77 30 2d 04 01 01 30 09 80 07 4e 32 32 30 31 21 30
0a 01 01 04 00 7e 18 a1 16 bc 14 04 0b 53 69 65 6d 65 6e 73 20 43 41 50 b0
05 82 03 35 35 39 -- Hipath 4000
void recvFastDataRequest(char* data, int len, CstaFastDataRequest2* csta){
   unsigned char tag;
   unsigned char taglen;
   tag=*(data++);
   taglen=*(data++);
   //char* Device = csta->device;
   //char* Manufacturer = csta->manufacturer;
   //char* ApplicationId = csta->applicationId;

   // 30 2d
   if(tag==0x30){
       // 04 01
       tag=*(data++);
       taglen=*(data++);
       //Prüfen ob ioRegisterReqId für diese Applikation gültig ist
       // 01 - ioRegisterReqId
       //if(_oss_dec_iint(0,&data,0,taglen)==ioRegisterReqId){
           data++;
           // 30 09
           tag=*(data++);
           taglen=*(data++);
           // 80 07 - DeviceID
           tag=*(data++);
           taglen=*(data++);
           _oss_dec_nstr(0, &(++data),0,taglen-3,csta->device, 10,0);
           data=data+2;
           // 0a 01
           tag=*(data++);
           taglen=*(data++);
           // DataPathType
           csta->DataPathType=_oss_dec_iint(0,&data,0,taglen);
           // 04 00
           tag=*(data++);
           taglen=*(data++);
           _oss_dec_nstr(0, &data,0,taglen, csta->ioData, 256,0);
           // 7e 18 - PrivateData
           tag=*(data++);
           taglen=*(data++);
           // a1 16
           tag=*(data++);
           taglen=*(data++);
           // bc 14
           tag=*(data++);
           taglen=*(data++);
           // 04 0b
           tag=*(data++);
           taglen=*(data++);
           // Manufacturer
           _oss_dec_nstr(0, &data,0,taglen, csta->manufacturer, 256,0);
           // b0 05
           tag=*(data++);
           taglen=*(data++);
           // 82 03
           tag=*(data++);
           taglen=*(data++);
           _oss_dec_nstr(0, &data,0,taglen, csta->applicationId, 256,0);
```

```
        //}
    }

}


//6B 0F 30 0D 80 02 00 E0 A1 07 30 05 80 03 31 30 30 63 07 30 05 80 03 31
30 30 4E 01 01 0A 01 16 A5 14 30 12 81 10 03 42 CF FA
void recvServiceInitiatedEvent(char* data,int
len,CstaServiceInitiatedEvent* csta) {
    unsigned char tag;
    unsigned char taglen;
    // 6B 9F
    tag=*(data++);
    taglen=*(data++);
    // 30 0D
    tag=*(data++);
    taglen=*(data++);
    // 80 02, CallId
    tag=*(data++);
    taglen=*(data++);
    csta->callId=_oss_dec_iint(0,&data,0,taglen);

    // A1 07
    tag=*(data++);
    taglen=*(data++);
    // 30 05
    tag=*(data++);
    taglen=*(data++);
    // 80 03
    tag=*(data++);
    taglen=*(data++);
    _oss_dec_nstr(0, &data,0,taglen, csta->device, 10,0);
}

//6B 0F 30 0D 80 02 00 E0 A1 07 30 05 80 03 31 30 30 63 07 30 05 80 03 31
30 30 4E 01 00 0A 01 30 A6 14 30 12 81 10 03 42 CF FA
void recvConnectionClearedEvent(char* data,int
len,CstaConnectionClearedEvent* csta) {
    unsigned char tag;
    unsigned char taglen;
    // 6B 9F
    tag=*(data++);
    taglen=*(data++);
    // 30 0D
    tag=*(data++);
    taglen=*(data++);
    // 80 02, CallId
    tag=*(data++);
    taglen=*(data++);
    csta->callId=_oss_dec_iint(0,&data,0,taglen);

    // A1 07
    tag=*(data++);
    taglen=*(data++);
    // 30 05
    tag=*(data++);
    taglen=*(data++);
    // 80 03
    tag=*(data++);
```

```
   taglen=*(data++);
   _oss_dec_nstr(0, &data,0,taglen, csta->device, 10,0);
}

void recvCallControlEvent(char* data,int len,CstaCallControlEvent* csta) {
   unsigned char tag;
   unsigned char taglen;
   //
   tag=*(data++);
   taglen=*(data++);
   csta->choice=tag;
   if (csta->choice==0xb0) recvServiceInitiatedEvent(data,len,&csta-
>specificEvent.serviceInitiatedEvent);
   if (csta->choice==0xa3) recvConnectionClearedEvent(data,len,&csta-
>specificEvent.connectionClearedEvent);
}

// B0 4A 6B 0F 30 0D 80 02 00 B6 A1 07 30 05 80 03 31 30 30 63 07 30 05 80
03 31 30 30 4E 01 01 0A 01 16 A5 14 30 12 81 10 03 94 7E FC
void recvPhysicalDeviceEvent(char* data,int len,CstaPhysicalDeviceEvent*
csta) {
   unsigned char tag;
   unsigned char taglen;
   //
   tag=*(data++);
   taglen=*(data++);

   if (tag==0xa1) recvButtonPressEvent(data,len,&csta-
>specificEvent.buttonPressEvent);
}

// A1 5B 02 02 03 71 02 01 15
// 30 52 55 02 00 00 A0 4C B0 4A 6B 0F 30 0D 80 02 00 B6 A1 07 30 05 80 03
31 30 30 63 07 30 05 80 03 31 30 30 4E 01 01 0A 01 16 A5 14 30 12 81 10 03
94 7E FC
void recvEvent(char* data,int len,CstaEvent* csta) {
   unsigned char tag;
   unsigned char taglen;
   //
   tag=*(data++);
   taglen=*(data++);
   tag=*(data++);
   taglen=*(data++);
   unsigned int crossRefId =_oss_dec_iint(0,&data,0,taglen);
   // A3 34 //PhysicalDeviceFeatureEvents
   tag=*(data++);
   len=*(data++);
   csta->choice=tag;
   if (tag==0xa0) recvCallControlEvent(data,len,&csta-
>eventSpecificInfo.callControlEvent);
   if (tag==0xa3) recvPhysicalDeviceEvent(data,len,&csta-
>eventSpecificInfo.physicalDeviceEvent);
}

// Verteiler für Dekodierungsfunktionen
void recvRoseRequest(char* data,int len) {
   CstaRequest csta;
   unsigned char tag;
   unsigned char taglen;
   tag=*(data++);
   taglen=*(data++);
   csta.invokeId=_oss_dec_iint(0,&data,0,taglen);
```

```
   tag=*(data++);
   taglen=*(data++);
   csta.opValue=_oss_dec_iint(0,&data,0,taglen);

   CString debug;
   debug.Format("%d %d",csta.invokeId,csta.opValue);

   switch (csta.opValue) {
   case 0xd3: //SystemStatus
       sendSystemStatusResponse(csta.invokeId);
       recvCstaRequest(&csta);
       debug.Format("SystemStatus");
       break;
   case 0x15: //ButtonPress
       debug.Format("ButtonPress");
       recvEvent(data,len,&csta.argument.event);
       recvCstaRequest(&csta);
       break;
   case 0x77: //FastData
       recvFastDataRequest(data,len,&csta.argument.fastDataRequest2);
       sendFastDataResponse(csta.invokeId);
       sendStartDataPathRequest(&csta.argument.fastDataRequest2);
       break;
   case 0x6e:;//StartDataPath
   recvStartDataPathRequest(data,len,&csta.argument.startDataPathRequest);
       recvCstaRequest(&csta);
       debug.Format("StartDataPath");
       break;
   case 0x70:;//SendData
       debug.Format("SendData");
       recvSendDataRequest(data,len,&csta);
       break;
   case 0x6f:;//StopDataPath
       recvStopDataPathRe-
quest(data,len,&csta.argument.stopDataPathRequest);
       recvCstaRequest(&csta);
       debug.Format("StopDataPath");
       break;
   case 0x4d:// SnapshotDeviceData
       recvCstaSnapshotDeviceDataRequest(data,len,
&csta.argument.snapshotDeviceDataRequest);
       break;
   default:
       recvCstaRequest(&csta);
   }
}



void recvLayer4(char* data,int len) {
   CString cstr;
   cstr.Format("%d",len);
   //PrintSysMsg("recvLayer4: "+cstr+": "+formatHex(data,len));
   char* start=data;
   unsigned char roseTyp=*(data++);
   char l4Len=*(data++);

   //len=len-int(data)+int(start);
   if (roseTyp==0xa1) recvRoseRequest(data,len);
   if (roseTyp==0xa2) recvRoseResponse(data,len);
   //if (roseTyp==0x61) recvRoseRequest(data,len);
}
```

```
char buffer[1024];
int inputPtr=0;

// Entry-Point TCP-Empfang
void recvTcp(char* data,int len) {
   memcpy(&buffer[inputPtr],data,len);
   inputPtr=inputPtr+len;

   while (1) {
       //RT:14-11:08 -- Request/Response in Byte 2
       int dataLen=buffer[1];
       if (inputPtr<dataLen+2) break;
       recvLayer4(&buffer[2],dataLen);
       memcpy(&buffer[0],&buffer[dataLen+2],inputPtr-dataLen-2);
       inputPtr=inputPtr-dataLen-2;
   }
}


void generateInvokeId(void)
{

g_InvokeId++;
if (g_InvokeId == 100){
   g_InvokeId = 1;
}

}

void   sendTcp(char* data,int len){
   //TODO: implement socket

   if(!g_socket.Send(data,len)){
       int serror=GetLastError( );
   }
}

void recvCstaRequest(CstaRequest* csta){
   char *s = (char *)malloc(1000);
   sprintf(s, "invokeId=%d, opValue=%d", csta->invokeId, csta->opValue);
   AfxMessageBox(s);
}

void recvCstaResponse(CstaRequest* csta){
   char *s = (char *)malloc(1000);
   sprintf(s, "invokeId=%d, opValue=%d", csta->invokeId, csta->opValue);
   AfxMessageBox(s);
}

void PDIAPI APITest(char test1, char test2) {

   CstaMakeCallRequest makeCallRequest;
   int i = sizeof(test1);
   int k = sizeof(test2);

   makeCallRequest.calledDirectoryNumber= &test1;
   makeCallRequest.callingDevice=&test2;


   sendMakeCallRequest(&makeCallRequest);
}
```

```
void PDIAPI closeConnection(){
   g_socket.Close();
}

int scanHex(CString string,char* p) {
   string+=" ";
   int a;
   for (a=0;a<string.GetLength()/3; a++) {
       CString zeichen=string.Mid(a*3,2)+" ";
       //if (zeichen==" ") return a;
       sscanf(zeichen,"%X ",p);
       p++;
   }
   return a;
}
```

. Die Header-Datei des CSTA - Treibers

```
/*************************************************************************
** Filename:      CstaApi.h                                             **
** Version        1.0                                                   **
** Autor:         Markus Franke, Rico Thomanek                          **
** Inhalt:        Anlegen aller benötigten Strukturen                   **
**                                                                      **
*************************************************************************/

#pragma once

//#include <afxwin.h>
#include <Afxsock.h>

#ifdef WIN32
  #define PDIAPI __declspec(dllexport)  WINAPI
#else
  #define PDIAPI __export __far __pascal
#endif



struct  CstaSetDisplayRequest {
char* device;
char* display;
};


struct CstaSystemStatusRequest {
   int SystemStatus;
   int SystemStatusArg;
};

struct CstaIoRegisterRequest {
   char* manufacturer;
   char* applicationId;
};

struct CstaIoRegisterResponse {
   int ioRegisterReqId;
};

struct CstaIoRegisterCancelRequest{
   char* applicationId;
};

struct CstaDataFastRequest {
   char* device;
   int DataPathType;
   char* ioData;
   char* manufacturer;
```

```
   char* applicationId;
   bool audi;
   bool fixed;
};

struct CstaFastDataRequest2 { // M.F. 10.2
char device [8];
int DataPathType;
char manufacturer[200];
char applicationId[10];
char ioData[200];
bool beep;
bool fixed;
};

struct CstaStartDataPathRequest {
   char* device;
};

struct CstaStartDataPathResponse {
   int ioCrossRefId;
};

struct CstaStopDataPathRequest {
   char* device;
   int ioCrossRefId;
};

struct CstaMonitorStartRequest {
   char* device;
   bool callCleard;
   bool CallConferenced;
   bool callConnectionCleared;
   bool callDelivered;
   bool callDiverted;
   bool callEstablished;
   bool callFailed;
   bool callBridged;
   bool callHeld;
   bool callQueued;
   bool callRetrieved;
   bool callDigitsDailed;
   bool callOffered;
   bool physicalButtonInfo;
   bool physicalButtonPress;
   bool physicalDisplayUpdated;
   bool physicalHookSwitch;
   bool physicalLampMode;
   bool logicalAutoAnswer;
};

struct CstaMonitorStartResponse{
int refIdMonitor;

};

struct CstaMonitorStopRequest{
int refIdMonitor;
char* device;
};

struct CstaMakeCallRequest {
```

```
   char* callingDevice;
   char* calledDirectoryNumber;
};

struct CstaButtonPressEvent {
   char device[8];
   int button;
};

struct CstaSendDataRequest {
   char* device;
   int crossRefId;
   char* ioData;
   char* manufacturer;
   char* applicationId;
   bool fixed;
   bool silent;
};

struct CstaConnectionClearedEvent {
   int crossRefId;
   int event;
   char device[8];
   int callId;
};

struct CstaServiceInitiatedEvent {
   int crossRefId;
   int event;
   char device[8];
   int callId;
};

struct CstaCallControlEvent {
   unsigned int choice;
   union {
       CstaConnectionClearedEvent connectionClearedEvent;
       CstaServiceInitiatedEvent serviceInitiatedEvent;
   } specificEvent;
};

struct CstaPhysicalDeviceEvent {
   unsigned int choice;
   union {
       CstaButtonPressEvent buttonPressEvent;
       //CstaPhysicalDeviceEvent physicalDeviceEvent;
   } specificEvent;
};


struct CstaEvent {
   unsigned int choice;
   union {
       CstaCallControlEvent callControlEvent;
       CstaPhysicalDeviceEvent physicalDeviceEvent;
   } eventSpecificInfo;
};

#pragma once
// MyThread

class MyThread : public CWinThread
```

```
{
   DECLARE_DYNCREATE(MyThread)

public:
   char daten [1024];
   int length;
   char device [8];

//protected:
public:
   MyThread();            // Dynamische Erstellung verwendet geschützten
Konstruktor
   virtual ~MyThread();

public:
   virtual BOOL InitInstance();
   virtual int ExitInstance();

protected:
   DECLARE_MESSAGE_MAP()
};


/*
class MyThread : public CWinThread
{
public:

   char daten[1024];
   int lenght;

};


struct CstaHoldCallRequest {   //hold Call

char device[8];
};


struct CstaClearConnectionRequest {

char device[8];
};


struct CstaAnswerCallRequest {
   int connId;
   char* device;

};
struct CstaParkCallRequest {
   char callId[8];
   char dailingdevice[8];

};
struct CstaJoinCallRequest {
   int callId;
   char dailingdevice[8];

};
struct CstaAlternateCallRequest {
```

```
   char heldCallId[8];
   char activeCallId[8];
};
struct CstaConferenceCallRequest {

   char heldCallId[8];
   char activeCallId[8];

};
struct CstaSnapshotCallRequest {

   char dialingId;
};
struct CstaSnapshotDeviceRequest {

   char* deviceId;
};
struct CstaSnapshotDeviceResponse {

   char Service_Cross_referenceID [4];
};

struct CstaSnapshotDeviceDataRequest{

   char callId[8];
   char ServiceCrossRefId[4];
   int connId;
   char* callState;
};


struct CstaMulticastDataRequest {

   int crossRefId;

};
//-------------------------------------------------------------------
struct CstaRequest {
   unsigned int invokeId;
   //int invokeId;
   unsigned int opValue;
   union {
        CstaFastDataRequest2 fastDataRequest2; //M.F. 10.2
        CstaIoRegisterResponse ioRegisterResponse;
        CstaIoRegisterCancelRequest ioRegisterCancelRequest;
        CstaDataFastRequest dataFastRequest;
        CstaStartDataPathRequest startDataPathRequest;
        CstaStopDataPathRequest stopDataPathRequest;
        CstaMonitorStartRequest monitorStartRequest;
        CstaMakeCallRequest makeCallRequest;
        //CstaSetDisplayRequest setDisplayRequest;
        //CstaButtonPressEvent buttonPressRequest;
        CstaMonitorStartResponse monitorStartResponse;
        CstaHoldCallRequest holdCallRequest;
        CstaSendDataRequest sendDataRequest;
//      CstaHoldCallRequest holdCallRequest;
        CstaClearConnectionRequest clearCallRequest;
        CstaParkCallRequest parkCallRequest;
        CstaJoinCallRequest joinCallRequest;
        CstaAlternateCallRequest alternateCallRequest;
        CstaConferenceCallRequest conferenceCallRequest;
```

```
        CstaSnapshotCallRequest snapshotCallRequest;
        CstaSnapshotDeviceRequest snapshotDeviceRequest;
        CstaMulticastDataRequest multicastDataRequest;
        CstaStartDataPathResponse startDataPathResponse;
        CstaSetDisplayRequest setDisplayRequest;
        CstaMonitorStopRequest monitorStopRequest;
        CstaSnapshotDeviceResponse snapshotDeviceResponse;
        CstaSnapshotDeviceDataRequest snapshotDeviceDataRequest;
        CstaAnswerCallRequest answerCallRequest;


        CstaEvent event;
    } argument;
};
```

```
//-----------------------------------------------------------------
int sendRoseRequest(char* data,int len,int op,int invokeId=0x77);
void sendRoseResponse(char* data,int len,int op,int invokeId=0x77);
void sendLayer4(char* data,int len);

// Entry-Points ASN1-Senden
void PDIAPI sendCstaSystemStatusRequest(CstaSystemStatusRequest* csta);
void PDIAPI sendCstaIoRegisterRequest(CstaIoRegisterRequest* csta);
void PDIAPI sendCstaIoRegisterCancelRequest (CstaIoRegisterCancelRequest*
csta);
void PDIAPI sendFastDataResponse(unsigned int invokeId);
void PDIAPI sendMakeCallRequest(CstaMakeCallRequest* csta);
void PDIAPI sendAARQ();
void PDIAPI sendMonitorStartRequest(CstaMonitorStartRequest* csta);
void PDIAPI sendSetDisplayRequest(char* device,char* display);
void PDIAPI sendCstaRequest(CstaRequest* csta);
void PDIAPI sendSystemStatusResponse(int invokeId);
void PDIAPI sendEscapeRequest();
void PDIAPI sendStartDataPathResponse(int invokeId,int crossRefId);
void PDIAPI sendSendDataRequest(CstaSendDataRequest* message);
void PDIAPI sendButtonPressRequest(char* device,int button);
void PDIAPI sendStartDataPathRequest(CstaDataFastRequest* csta);
void PDIAPI sendSetDisplayRequest(CstaSetDisplayRequest* csta);
void PDIAPI sendFastDataRequest(CstaDataFastRequest* csta);
void PDIAPI sendAlternateCallRequest(CstaAlternateCallRequest *csta);
void PDIAPI sendConferenceCallRequest(CstaConferenceCallRequest *csta);
void PDIAPI sendHoldCallRequest(CstaHoldCallRequest *csta);
void PDIAPI sendClearConnectionRequest (CstaClearConnectionRequest *csta);
void PDIAPI sendMonitorStopRequest(CstaMonitorStopRequest *csta);
void PDIAPI sendStopDataPathRequest(CstaStopDataPathRequest *csta);
//------- eigene-------------


void PDIAPI sendAnswerCallRequest (CstaAnswerCallRequest *csta);
void PDIAPI sendParkCallrequest (CstaParkCallRequest *csta);
void sendJoinCallRequest ();
void sendAlternateCallRequest ();
void sendConferenceCallRequest ();
void sendSnapshotCallRequest ();
void PDIAPI sendSnapshotDeviceRequest (CstaSnapshotDeviceRequest *csta);
void sendMulticastDataRequest ();
```

```
//------------------------------

// Entry-Point TCP-Empfang
void recvTcp(char* data,int len);
void recvLayer4(char* data,int len);
void recvRoseRequest(char* data,int len);
void sendTcp(char*,int);
// Exit-Point ASN1-Empfang
void recvCstaRequest(CstaRequest* csta);
void recvCstaResponse(CstaRequest* csta);

// Dekoderfunktionen
void recvCstaIoRegisterResponse(char* data,int len,CstaIoRegisterResponse*
csta,int invokeId);
void recvFastDataRequest(char* data, int len, CstaDataFastRequest* csta);
void recvSendDataRequest(char* data,int len,CstaRequest* csta,int in-
vokeId);
void recvStartDataPathRequest(char* data,int len,CstaStartDataPathRequest*
csta);
void recvStopDataPathRequest(char* data,int len,CstaStopDataPathRequest*
csta);
void recvButtonPressEvent(char* data,int len,CstaRequest* csta);
void recvStartDataPathResponse(char* data, int len, CstaStartDataPathRe-
sponse* csta);
void recvMonitorStartResponse(char* data, int len, CstaMonitorStartRe-
sponse* csta);
void recvSnapshotDeviceResponse(char* data, int len, CstaSnapshotDeviceRe-
sponse* csta);
void recvCstaSnapshotDeviceDataRequest(char* data, int len, CstaSnapshotDe-
viceDataRequest* csta);


void generateInvokeId(void);
int scanHex(CString string,char* p);
```

Anlage B: Hilfsfunktionen

Asn1Impl.cpp

Die Cpp-Datei mit den Hilfsfunktionen

```
/*************************************************************************
** Filename:      Asn1Impl.cpp                                          **
** Version        1.0                                                   **
** Autor:         R. Thomanek, F. Neuner                                **
** Inhalt:        Hilfsfunktionen zur Encodierung und Decodierung       **
**                                                                      **
*************************************************************************/


//----- coding of integer variable length
long _oss_encd_int(struct ossGlobal *_g,char** pbuf,long*, char,long i) {
   int len=1;
   //if(_g->decodingFlags & RESERVED_FLAG2) len=2;
   if (i>127 || i<-128) len=2;
   if (i>32767 || i<-32768) len=4;
   char* buf=*pbuf;
   buf=buf-len;

   char* indata=(char*)&i;
   for (int c=0;c<len;c++) buf[c]=indata[len-1-c];

   *pbuf=buf;
#ifdef debug_ossber
   printf("---------- _oss_encd_int nicht getestet !!! ----------\n");
#endif
   return len;
}

//----- coding of lenght, always one byte
long _oss_encd_length(struct ossGlobal *, char** pbuf, long*,char, long un-
signed _data_len) {
   int len=1;
   char* buf=*pbuf;
   buf=buf-len;
   buf[0]=(char)_data_len;
   *pbuf=buf;
#ifdef debug_ossber
   printf("---------- _oss_encd_length nicht getestet !!! ----------\n");
#endif
   return len;
}

//----- coding of tags
long _oss_encd_tag(struct ossGlobal* , char** pbuf, long*,char,unsigned
short tag,char _constructed) {
   int len=1;
   // default tag encoding
```

```
   // low byte and high byte are added
   //printf("%x %d\n",tag,_constructed);
   unsigned short ctag=tag+tag/256;


   char* buf=*pbuf;
   buf=buf-len;
   if (len==1) *((unsigned char*)buf) =(unsigned char)ctag;
   if (len==2) *((unsigned short*)buf)=ctag;
   *pbuf=buf;


   #ifdef debug_ossber
       printf(" _oss_encd_tag ctag: %x tag: %x\n",ctag,tag);
   #endif
   return len;
}

// wird verwendet
void _oss_enc_push(struct ossGlobal *g, void* temp) {
#ifdef debug_ossber
   printf("---------- _oss_enc_push nicht getestet !!! Achtung kein Stack!
----------\n");
#endif
}

// wird verwendet
void* _oss_enc_pop(struct ossGlobal *g) {
#ifdef debug_ossber
   printf("---------- _oss_enc_pop nicht getestet !!! ----------\n");
#endif
   return NULL;
}

//----- coding of variable length character string
long _oss_encd_nstr(struct ossGlobal* _g, char** pbuf, long*,char, char* c,
long i) {
   int len=strlen(c);
   char* buf=*pbuf;
   buf=buf-len;
   strncpy(buf,c,len);
   *pbuf=buf;
#ifdef debug_ossber
   printf("---------- _oss_encd_nstr nicht getestet ----------\n");
#endif
   return len;
}

long _oss_encd_uint(struct ossGlobal *, char** pbuf, long*,char, long un-
signed i) {
   int len=1;
   char* buf=*pbuf;
   buf=buf-len;
   *((unsigned char*)buf)=i;
   *pbuf=buf;
#ifdef debug_ossber
   printf("---------- _oss_encd_uint nicht getestet ----------%x\n",buf);
#endif
   return len;
}
```

```
long _oss_encd_voct(struct ossGlobal* _g, char** pbuf, long*,char, void*
oct, char off,long len) { //2, 8);
   len=*((short*)oct);
   *pbuf=*pbuf-len;
   void* buf=*pbuf;
   memcpy((unsigned char*)buf,(unsigned char*)oct+off,len);
#ifdef debug_ossber
   printf("---------- _oss_encd_voct nicht getestet ----------\n");
#endif
   return len;

}

// verwendet
/*
long _oss_encd_gtime(struct ossGlobal *, char**, long*,char, Generalized-
Time*) {
#ifdef debug_ossber
   printf("---------- _oss_encd_gtime nicht implementiert !!! ----------
\n");
#endif
   return 0;
}
*/

// verwendet
long _oss_encd_bool(struct ossGlobal*, char** pbuf, long*,char, char value)
{
#ifdef debug_ossber
   printf("---------- _oss_encd_bool nicht getestet !!! ----------\n");
#endif
   int len=1;
   char* buf=*pbuf;
   buf=buf-len;
   if (value) value=0xff;
   *((unsigned char*)buf)=value;
   *pbuf=buf;
   return len;
}

// [ASN1:S.266]
// indata ist ein unsigned numerischer Typ, die Bytes mit den MSB
(bits_used )müssen kopiert werden
long _oss_encd_pbit(struct ossGlobal *_g, char** pbuf, long*,char,void* in-
data,long bits_used) {
   unsigned char bytes_used=(bits_used+7)/8;
   int len=bytes_used+1;
   *pbuf=*pbuf-len;
  unsigned char* outbuf=(unsigned char*)(*pbuf);
  unsigned char* inbuf=(unsigned char*)indata;

  *outbuf=8*bytes_used-bits_used;
  int bytes_len=bytes_used;
  //if (bytes_used>2) bytes_len=4;

   for (int i=1;i<=bytes_used;i++) {
       *(++outbuf)=*(inbuf+bytes_len-i);
   }
   #ifdef debug_ossber
       printf(" _oss_encd_pbit bits_used: %d in_data:
%x\n",bits_used,indata);
   #endif
```

```
   return len;
}

// CSTA
long _oss_encd_uoct(struct ossGlobal *, char**, long*,char,void*,
char,long) {
#ifdef debug_ossber
   printf("---------- _oss_encd_uoct nicht implementiert !!! ----------
\n");
#endif
   return 0;
}

// nicht verwendet
long _oss_encd_lsobjid(struct ossGlobal *, char**, long*,char, void*,long)
{
   return 0;
}

// nicht verwendet
/*
long _oss_encd_utime(struct ossGlobal *, char**, long*,char, UTCTime*) {
   return 0;
}
*/

// nicht verwendet
long _oss_encd_uany(struct ossGlobal *, char**, long*,char, void*) {
   return 0;
}



//-------------------------------------------------------------------------
----------------------------
// decoding
/*
void _oss_dec_error(struct ossGlobal *_g, char, _err_index error,long par)
{
   ossPrint(_g,"dec_error=%d tag=%d",error,par);
   #ifdef debug_ossber
       printf("------------- _oss_dec_error ------------ \n");
   #endif
}
*/

long _oss_dec_length(struct ossGlobal *_g, char** pbuf,long*) {
   int len=1;
   char* buf=*pbuf;
   *pbuf=buf+len;
   #ifdef debug_ossber
       printf("------------- _oss_dec_length ------------
länge=%d\n",*((unsigned char*)buf));
   #endif
   return *((unsigned char*)buf);
}

// decoding of short int
short _oss_dec_sint(struct ossGlobal*, char** pbuf,long* plong,long len) {
   char* buf=*pbuf;
   *pbuf=buf+len;
```

```
   short sint=0;
   if (len==1) sint=*((char*)buf);
   if (len==2) sint=*((short*)buf);
#ifdef debug_ossber
   printf("_oss_dev_iint len=%d return=%d plong=%d nicht getes-
tet\n",len,sint,*plong);
#endif
   return sint;
}

// decoding of vartiable length int
int _oss_dec_iint(struct ossGlobal*, char** pbuf,long* plong,long len) {
   char* buf=*pbuf;
   *pbuf=buf+len;
   int iint=0;

   char* outdata=(char*)&iint;
   for (int c=0;c<len;c++) outdata[len-1-c]=buf[c];

#ifdef debug_ossber
   printf("_oss_dev_iint len=%d return=%d plong=%d nicht getes-
tet\n",len,iint,*plong);
#endif
   return iint;
}

//
void* _oss_dec_getmem(struct ossGlobal *_g,long strgroe,char bufpos) {
#ifdef debug_ossber
   printf("---------- _oss_dec_getmem ---------struktur-Groesse:%d puffer-
pos:%c return:%x\n",strgroe,bufpos,(((_EncDecGlobals*)(_g->encDecVar))-
>_oss_outbufpos));
#endif
   return NULL;//(((_EncDecGlobals*)(_g->encDecVar))->_oss_outbufpos;
}

// ???
/*
void _oss_releaseMem(struct ossGlobal *,_mem_array*) {
#ifdef debug_ossber
   printf("---------- _oss_releaseMem nicht implementiert !!! ----------
\n");
#endif
}
*/

// decoding of strings
void _oss_dec_nstr(struct ossGlobal *_g, char** pbuf,long*,long len, char*
dst, long max, char) {
   char* buf=*pbuf;
   *pbuf=buf+len;
   strncpy(dst,buf,len);
   dst[len]=0;
#ifdef debug_ossber
   printf("---------- _oss_dec_nstr nicht getestet ----------\n");
#endif
}

// decoding of strings
unsigned int _oss_dec_uiint(struct ossGlobal *_g, char** pbuf,long*,long
len) {
   char* buf=*pbuf;
```

```
   *pbuf=buf+len;
   unsigned int uint=0;
   if (len==1) uint=*((char*)buf);
   if (len==2) uint=*((unsigned int*)buf);

#ifdef debug_ossber
   printf("---------- _oss_dec_uiint nicht getestet ----------\n");
#endif
   return uint;
}

// verwendet
void _oss_dec_voct(struct ossGlobal *, char**,long*,long,
void*,char,long,char _constructed) {
#ifdef debug_ossber
   printf("---------- _oss_dec_voct Nicht implementiert !!! ----------\n");
#endif
}

// verwendet
/*
void _oss_dec_gtime(struct ossGlobal *, char**,long*,long, Generalized-
Time*) {
#ifdef debug_ossber
   printf("---------- _oss_dec_gtime Nicht implementiert !!! ----------
\n");
#endif
}
*/

// verwendet
long _oss_dec_llint(struct ossGlobal *, char**,long*,long) {
#ifdef debug_ossber
   printf("---------- _oss_dec_llint Nicht implementiert !!! ----------
\n");
#endif
   return 0;
}

// verwendet
BOOL _oss_dec_bool(struct ossGlobal *, char**,long*,long) {
#ifdef debug_ossber
   printf("---------- _oss_dec_bool Nicht implementiert !!! ----------\n");
#endif
   return 0;
}

// verwendet
void _oss_dec_pbit(struct ossGlobal *, char**,long*,long, void*, long un-
signed,char,char) {
#ifdef debug_ossber
   printf("---------- _oss_dec_pbit Nicht implementiert !!! ----------\n");
#endif
}

void _oss_dec_tag(struct ossGlobal *_g, char** pbuf, long*, unsigned short*
tag, BOOL*) {
   int len=1;
   char* buf=*pbuf;
   unsigned char c1=*((unsigned char*)buf);  //invokeClass
   unsigned char c2=*((unsigned
char*)buf+4);//callHandlingService,applControlService
```

```
   unsigned short highbyte=(c1 & 0xC0)*256;
   unsigned short lowbyte=c1 & 0x1F;
   *tag=lowbyte + highbyte;
  //callHandlingServices, siehe [ACL-C:S.198]
  if (c1==0x66 && c2==0x29) *tag=0x4021; // AlternateCall-Response
  if (c1==0x66 && c2==0x2B) *tag=0x4023; // AnswerCall-Response
  if (c1==0x66 && c2==0x30) *tag=0x4095; // CallBack-Response
  if (c1==0x66 && c2==0x3C) *tag=0x4025; // ClearConnection-Response
  if (c1==0x66 && c2==0x35) *tag=0x4089; // ConnectTimeslot-Response
  if (c1==0x66 && c2==0x2A) *tag=0x4029; // ConferenceCall-Response
  if (c1==0x66 && c2==0x1E) *tag=0x402B; // ConsultCall-Response
  if (c1==0x66 && c2==0x13) *tag=0x4048; // ContinueCall-Response
  if (c1==0x66 && c2==0x34) *tag=0x408B; // DeflectCall-Response
  if (c1==0x66 && c2==0x2E) *tag=0x4077; // DialDigits-Response
  if (c1==0x66 && c2==0x14) *tag=0x4031; // DivertCall-Response
  if (c1==0x66 && c2==0x2F) *tag=0x4079; // GenerateDigits-Response
  if (c1==0x66 && c2==0x33) *tag=0x408D; // GenTelephonyTone-Response
  if (c1==0x66 && c2==0x1F) *tag=0x404C; // HoldCall-Response
  if (c1==0x66 && c2==0x2D) *tag=0x4075; // InvokeFeature-Response
   if (c1==0x66 && c2==0x0A) *tag=0x402D; // MakeCall-Response
   if (c1==0x66 && c2==0x32) *tag=0x408F; // MakePredictiveCall-Response
   if (c1==0x66 && c2==0x3D) *tag=0x402F; // ReconnectCall-Response
   if (c1==0x66 && c2==0x36) *tag=0x4091; // ReconnectTimeslot-Response
   if (c1==0x66 && c2==0x12) *tag=0x404A; // RejectCall-Response
   if (c1==0x66 && c2==0x20) *tag=0x404E; // RetrieveCall-Response
   if (c1==0x66 && c2==0x2C) *tag=0x406B; // SingleStepTransferCall-
Response
   if (c1==0x66 && c2==0x31) *tag=0x4093; // SendUserInformation-Response
   if (c1==0x66 && c2==0x15) *tag=0x4033; // TransferCall-Response
   if (c1==0x66 && c2==0x19) *tag=0x4037; // Acd-Queue-Status-Response

#ifdef debug_ossber
   printf("_oss_dec_tag asn=%x oss=%x\n",c1,*tag);
#endif
   //callStatusServices
   //applControlServices
   *pbuf=buf+len;
}

// CSTA
void _oss_dec_nstr_ptr(struct ossGlobal *, char**,long*,long, char,
char**,long,char) {
#ifdef debug_ossber
   printf("------------- _oss_dec_nstr_ptr -------------- nicht implemen-
tiert\n");
#endif
}

// CSTA
void _oss_dec_uoct(struct ossGlobal *, char**,long*,long, char,
void*,char,long,char) {
#ifdef debug_ossber
   printf("------------- _oss_dec_uoct -------------- nicht implemen-
tiert\n");
#endif
}

// nicht verwendet
void _oss_dec_lsobjid(struct ossGlobal *, char**,long*,long, char, void**,
long) {
}
```

```
// nicht verwendet
/*
void _oss_dec_utime(struct ossGlobal *, char**,long*,long, UTCTime*) {
}
*/

// nicht verwendet
void _oss_dec_uany(struct ossGlobal *, char**,long*,long, char, void*) {
}

void _oss_set_outmem_d(struct ossGlobal *,long,long*,char**) {
#ifdef debug_ossber
   printf("---------- _oss_set_outmem nicht implementiert !!! ----------
\n");
#endif
}

// Konverter sendet immer lokalen Op-ID, deswegen nicht implementiert
/*
long _oss_encd_eobjid(struct ossGlobal *g, _std_parms_def,void *data, long
size_c) {
#ifdef debug_ossber
   printf("---------- _oss_set_eobjid nicht implementiert !!! ----------
\n");
#endif
}
*/

// CSTA
long _oss_encd_opentype(struct ossGlobal *g, char** pbuf,long*, char,void
*_data) {
return 0;
#ifdef debug_ossber
   printf("---------- _oss_encd_opentype nicht getestet ----------\n");
#endif
   return 0;
}
```

Literaturverzeichnis

[1] /Georg/	O. Georg: Telekommunikationstechnik Handbuch für Praxis und Lehre, Springer Verlag, 2. Auflage, ISBN: 3-540-66845-4
[2]/Aeschbacher/	Programmieren in C,C++, Software-Schule Schweiz, Roland Aeschbacher Seite 139
[3] /Siegmund/	G. Siegmund, Grundlagen der Vermittlungstechnik, R. v. Deckers Verlag, 2. Auflage, ISBN: 3-7685-4892-9
[4]Praktikumsarbeit	Praktikumsarbeit Matthias Baier 2009
[5] /Larmouth/	J. Larmouth: ASN.1 Complete, PDF
[6]/ NET/	Fachzeitschrift NET 06/07 Gruppenarbeit – CTI-Funktionalität für effektive Kommunikation
[7]WEB	http://telecom.htwm.de/telecom/praktikum/CTI_TAPI/CTI_TAPI.htm
[8]WEB	http://telecom.htwm.de/asn1/index.htm
[9]WEB	http://se.cs.uni-magdeburg.de/tutorial/Anwendungsfalldiagramm.htm
[10]WEB	http://se.cs.uni-magdeburg.de/tutorial/Zustandsdiagramm.htm
[11] WEB	http://se.cs.uni-magdeburg.de/tutorial/Aktivitdtsdiagramm.htm
[12]WEB	http://www.oss.com/products/tools.html

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

<Unterschrift>

Mittweida, Datum