Integration des Modbus-RTU-Protokolls in einem ressourcenschonenden System

BACHELORARBEIT IM RAHMEN DES
BACHELORSTUDIENGANGS INFORMATIK
SOFTWARE AND INFORMATION ENGINEERING

Dipl.-Ing Armin Simma
Fachhochschule Vorarlberg

Michael Theurl
Firma smog.at GmbH

Vorgelegt von

Martin Münch
1110247032

Dornbirn, 8. Februar 2014

Zusammenfassung

Die aufgabenstellung des Berufspraktikums bestand darin, das Potential von verschieden System On A Chip (SOC)-Modellen in Bezug zu den Kommunikationsprotokollen Modbus, Radio-Frequency Identification (RFID) und Universal Mobile Telecommunications System (UMTS) zu evaluieren. Jedes der drei Protokolle sollte in einem eigenständigen Projekt realisiert und dokumentiert werden. Diese Arbeit nimmt in erster Linie Bezug auf das Projekt Visual Energy, welches über das Modbus-Protokoll kommunizieren soll. Das Ziel bestand darin, mittels eines System On A Chip (SOC) die Messwerte von zwei definierten Energiezähler-Modellen der Firma Saia-Burgess auszulesen. Auf Grund der durchgeführten State of the Art-Analyse fiel hardware-seitig die Entscheidung zu Gunsten des Gnublin-Boards aus. Im Softwarebereich kristallisierte sich die in „C“ geschriebene Bibliothek Libmodbus heraus, welche in einem Python3 Programm verwendet werden sollte. Aufbauend auf den analysierten Ergebnissen wurden Überlegungen bezüglich der Softwarearchitektur erläutert und anschließend in ein Konzept-Modell umgesetzt. Bei der Implementierung wurde der Schwerpunkt auf die Kommunikation zwischen den Energiezählern und dem SOC gesetzt. Als Ergebnis des Projektes ist das in Python3 entwickelte Programm Visual Energy entstanden. Für die Kommunikation mit den Energiezählern wurde die, auch eigenständig nutzbare, Applikation ComModbus entwickelt und in das Programm Visual Energy eingebettet.

Abstract

The main task of the internship consisted in evaluating the potential of different System On A Chip concepts, regarding the communication protocols Modbus, RFID and UMTS. Each of these three protocols has been realised and documented in a distinct project. This work mainly focuses on the project Visual Energy, which is supposed to communicate via Modbus. The goal was to read measured data from two different energy meters by Saia-Burgess. Based on the accomplished state of the art analysis, the Gnublin Board has been selected as the hardware component. The library Libmodbus, which is written in C, has been chosen to be used in a Python 3 program. Based on the results of the analysis the software architecture has been outlined and concept model. The focus oft he implementation clearly lies on the communication of the SOC and the energy meters. The result of the project is the program Visual Energy, developed in Python 3. For the communication with the energy meters the application ComModbus has been developed. It has been integrated in the program Visual Energy but is also independently utilizable.

Eidesstattliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Inhaltsverzeichnis

1 Einführung **3**

2 State of the Art **6**
2.1 Rahmenbedingungen 6
2.1.1 Hardware-Anforderung 6
2.1.2 Software-Anforderung 7
2.2 Übersicht der Technologien 7
2.2.1 Hardware 7
2.2.2 Software 9

3 Analyse **17**
3.1 Auswahl der Technologien 17
3.1.1 Eingesetzte Hardware 17
3.1.2 Eingesetzte Software 18
3.2 Spezifikation 20

4 Entwicklung **21**
4.1 Überlegungen zur Softwarearchitektur 21
4.1.1 Entwurf des Modells 22

5 Implementierung **25**
5.1 Aufbau der Entwicklungsinfrastruktur 25
5.2 Allgemeines Vorgehen bei der Implementierung 25
5.3 Implementierung der Kommunikations-Schicht 26
5.3.1 Energiezähler spezifische Informationen 27
5.3.2 Planung des Modbus-Moduls 28
5.3.3 Implementierung ComModbus 28
5.3.4 Anwendungsfall ComModbus 30
5.3.5 Realisierung von ComModbus 32
5.3.6 Einbindung von ComModbus in der Kommunikationsschicht 34

6 Evaluation **36**
6.1 Ergebnisse 36
6.1.1 Tests & Benchmarks 37
6.2 Ausblick 39
6.3 Persönliche Meinung 40

A Diagramme und Bilder **46**

B Befehlssatz ComModbus **54**
B.1 Konfigurationen 54
B.2 Operationen 57

Kapitel 1

Einführung

Diese Arbeit soll die Erkenntnisse dokumentieren welche ich im Zuge des Berufspraktikums bei der Firma **smog.at** GmbH im Sommer 2013 erwerben durfte. Die Firma **smog.at** GmbH ist ein junges, innovatives und aufstrebendes Unternehmen mit Hauptfirmensitz in Lauterach. Zu ihren Tätigkeitsfeld zählen unter anderem: Netwerk- und Kommunikations-Technologien, Sicherheits- und Risiko-Management sowie Zertifizierung von IT-Umgebung.
Der Hauptaufgabenbereich meines Praktikums bestand darin, verschiedene Kommunikationskanäle wie beispielsweise Universal Mobile Telecommunications System (UMTS), Radio-Frequency Identification (RFID) und Modbus auf verschiedenen System On A Chip (SOC)-Modellen zu implementieren und das Vorgehen reproduzierbar zu dokumentieren. Die folgenden Abschnitte stellen eine Übersicht der verschiedenen, von mir bearbeiteten Projekte da.

Datenverbindung via UMTS

Der Inhalt dieses Projektes ist es, eine Datenverbindung für ein bestehendes Server-Monitoring-Systems zu realisieren. Der Nutzen des Systems besteht darin, den Zustand der registrierten Dienste, wie zum Beispiel der eines Webservers, zu überwachen. Sollte bei einer registrierten Komponente ein kritischer Zustand eintreten, besteht die Möglichkeit, auf verschiedene Wegen die definierten Verantwortlichen zu benachrichtigen.
Hierbei liegt die Aufgabenstellung nicht darin, die Überwachungssoftware selbst zu entwickeln, sondern vielmehr die Einbettung und Anbindung eines UMTS-Modems an das Betriebssystem und dessen Dienste sicherzustellen. Als Grundlage für diese Aufgabe dient ein SOC. Software-seitig wird CentOS als Betriebssystem verwendet.

Folgende Punkte sind im Rahmen der Spezifikation vorgegeben:
Um die Verfügbarkeit zu erhöhen soll neben den Standartkonfigurationen, wie zum Beispiel Zugang zum Netzwerk, eine redundante Netzwerkschnittstelle mit Hilfe eines UMTS-Modem realisiert werden. Diese soll des weiteren auch für den automatisierten Versand von SMS, sowie im Falle eines Netzwerkausfalls als Zugriffsmöglichkeit für Wartungsarbeiten verwendet werden. Zusätzlich darf die UMTS-Verbindung während des automatisierten Versands von SMS nicht unterbrochen werden.
Für den sicheren Zugang zu geschlossenen Netzwerken soll OpenVPN eingesetzt werden. Des Weiteren sollen Wartungszugriffe auf das Gerät über das Secure Shell (SSH)-Protokoll erfolgen.

Kommunikation via RFID

In diesem Projekt soll - mit minimaler Hardware-Ressourcen - eine Grundlage für die Entwicklung und Einbettung eines RFID-Shield in einem Linux Betriebssystem realisiert sowie für zukünftige Einsatzmöglichkeiten dokumentiert werden. Ein weiteres Ziel besteht darin, dass die Umsetzung bis auf die Limitierung, dass ein Linux-Betriebssystem eingesetzt werden soll, Plattform unabhängig ist. Dies soll die Grundlage dafür bilden, weitere Applikationen zu entwickeln, welche auf den dokumentierten Konfigurationen aufbauen.
Neben einem SOC wird ein „Philips PN532/C1“ als RFID-Komponente eingesetzt. Als Betriebssystem wird eine für ARM-Architekturen portierte Debian-Distribution verwendet. Die Kommunikation wird über die frei erhältliche `libnfc`-Bibliothek realisiert.

Visual Energy

Bei dem Projekt Visual Energy handelt es sich um eine Konzeptlösung, bestehend aus Hard- und Softwarekomponenten.
In erster Linie soll der Funktionsumfang das Auslesen und Speichern von gemessenen Verbrauchswerten umfassen. Dabei werden die Messwerte eines Energiezählers, wie zum Beispiel die aktuelle Spannung, ausgelesen und weiter verarbeitet.
Das Ziel ist es, die ermittelten Daten dem/der NutzerInn auf eine einfache Art zur Verfügung zu stellen. Die Verwendung von Visual Energy gibt den AnwenderInnen einen transparenten Einblick in den persönlichen Energieverbrauch. Durch eine Präsentation der Verbrauchsanalyse soll eine Sensibi-

lisierung stattfinden und im besten Fall gar eine Bewusstseinsbildung zum nachhaltigen und verantwortungsvollen Umgang mit der Ressource Energie anstoßen.

Da es denn Rahmen sprengen würde, alle drei Projekte ausführlich zu beschreiben nimmt diese Arbeit in erster Linie Bezug auf das Projekt Visual Energy.

Kapitel 2

State of the Art

2.1 Rahmenbedingungen

Eine große Herausforderung des Projektes Visual Energy besteht in der kreativen Freiheit bei der Auswahl der Technologien. Diese Freiheit kommt durch die verhältnismäßig geringen Rahmenbedingungen an das Projekt zustande. Für eine besser lesbare Struktur sind die Anforderungen in die Kategorien Soft- und Hardware unterteilt.

2.1.1 Hardware-Anforderung

Gewisse Hardwarekomponenten wurden vor dem Projektstart explizit definiert, andere wurden nur grob umrissen.
Unter den unbedingt zu verwendenden Hardwarekomponenten befinden sich ausschließlich die zwei Energiezähler der Firma Saia-Burgess. Dabei handelt es sich zum einem um das dreiphasige Modell: Saia-Burgess ALE3 (vgl. *Datenblatt Modbus ALE3* 2010) und zum anderem um das einphasige Gerät: Saia-Burgess ALD1 (vgl. *Datenblatt Modbus ALD1* 2011).
Beide Modelle kommunizieren mithilfe des Modbus-Protokoll . Als Recheneinheit soll ein SOC eingesetzt werden. Diese Bauart zeichnet sich durch die geringe Bauröße, den niedrigen Energieverbrauch sowie die günstigen Anschaffungskosten aus. Ein spezieller Hersteller, beziehungsweise Modell, wurde nicht definiert. Schlussendlich werden zwei Bedingungen an den SOC gestellt. Zum einem muss das Gerät netzwerkfähig sein und zum anderem über einen USB-Schnittstelle verfügen.

2.1.2 Software-Anforderung

Die Anforderungen an die Software leiten sich zum größten Teil von den Hardwarekomponenten ab. Durch die Verwendung der Energiezähler ALE3 und ALD1 ist die Verwendung des Modbus-Protokoll unabdingbar. Des Weiteren ist das verwendete Betriebssystem teilweise von der gewählten Hardwarearchitektur abhängig.

2.2 Übersicht der Technologien

Das Ziel dieses Kapitel ist es, die für das Projekt relevanten Technologien vorzustellen, die sich durch die State of the Art – Recherche heraus kristallisiert haben. Diese bilden die Grundlage für die Auswahl der Technologien, die schlussendlich im Projekt eingesetzt werden.

2.2.1 Hardware

Wie im Kapitel Hardware-Anforderungen schon beschrieben wurde, ist keine genauere Spezifikation für die Hardware des Projektes Visual Energy-definiert. Da als Anforderungsprofil entschieden wurde, dass ein SOC zum Einsatz kommen soll, werden an dieser Stelle ausschließlich eine Auswahl an relevanten SOC' s vorgestellt.

Hardware-Schnittstellen

Das Anforderungsprofil an die Hardwareschnittstellen ist sehr minimalistisch und lässt sich auf exakt zwei Schnittstellen herunterbrechen. Zum einem ist dies die Anbindung an die Energiezähler und zum anderem die Verfügbarkeit einer kabelgebundenen Netzwerkkonnektivität. Für die Realisierung der Bedingung wurden folgende Schnittstellen definiert: Ethernet für den Netzwerkzugang und ein handelsüblicher USB-Anschluss für die Verbindung zu den Energiezählern.

System On A Chip

Unter einem System On A Chip oder Einzelplatinen-Computer versteht man ein funktionsfähiges System, dessen Kernkomponente ausschließlich auf einer Leiterplatte montiert sind. Durch die einfache Konstruktionsform und das relativ minimalistische Leistungsniveau der Hardwarekomponenten sind Einzelplatinen-Computer sehr günstig in der Anschaffung. Ein weiterer Vorteil, neben dem geringen Preis, stellen die programmierbare Schnittstelle,

die sogenannte General Purpose Input/Output (GPIO) dar. Über die GPIO-Schnittstelle können einzelne Pins als Ein- beziehungsweise Ausgänge definiert werden. Aus diesem Grund eignen sich Einzelplatinen-Computer ausgezeichnet für den schnellen, risikoarmen und kostensparenden Aufbau von Prototypen.

Raspberry Pi

Der Raspberry Pi wird von der in England als wohltätig eingetragenen Stiftung: „Raspberry Pi Foundation“, mit der Zielsetzung, das Studium der Informatik zu fördern, vertrieben. Die Konzeption des Raspberry Pi wurde für eine Ausbildungs- und Entwicklungsplattform ausgelegt. Für die Analyse stand das Modell: „Raspberry Pi Modell B“ zur Verfügung. Der Rasperry Pi ist mit einem 700MHz starken ARM11-Prozessor und 512MB RAM ausgestattet. Der Raspberry Pi verfügt über reichlich Hardwareschnittstellen. Neben einer Ethernet-Schnittstelle sind zwei USB-Anschlüsse, ein 3,5mm Klinkenstecker, ein HDMI-Anschluss, 16 GPIO-Pins sowie einen Steckplatz für eine SD-Karte, auf der sich das Betriebssystem befindet, vorhanden. Die Stromversorgung erfolgt über den Micro USB-Anschluss. Abgesehen von einem USB- und dem Ethernet-Anschluss sind die Schnittstellen für das Projekt nicht weiter relevant.
Leider hat der Raspberry Pi keinen direkten Konsolen-Anschluss für den externen Zugriff. Wenn ein Zugriff auf den Raspberry Pi von einem Entwicklungs-PC aus möglich sein soll, so muss dies über das SSH-Protokoll oder über die GPIO-Anschlüsse mit einem speziellen Konsolen-Kabel erfolgen. (vgl. Raspberry Pi Foundation 2014a; und Raspberry Pi Foundation 2014b)
Der Raspberry Pi erfüllt das Anforderungsprofil vollständig.

Gnublin

Das Gnublin Board wurde laut Hersteller als Entwicklungsplattform ausgelegt. Für die Evaluierung stand das Modell: „Gnublin-LAN“ zur Verfügung. Mit dem 180 MHz Advanced RISC Machines (ARM)9-Prozessor und dem 32 MB Synchronous Dynamic Random Access Memory (SDRAM) hat das Gnublin-Board von den zur Verfügung stehenden SOC' s die geringsten Hardwareressourcen. Über den Gnublin Connector können weitere Gnubiln-Hardware-Module oder externe Peripherie angeschlossen werden. Der Gnublin bietet den Vorteil, dass die angeschlossen Module über die Gnublin-API leicht und schnell angesprochen werden können. Der Gnublin verfügt über einen integrierten Konsolen Anschluss der durch einen USB auf RS232-Konverter realisiert wurde. Durch den Konsolen Anschluss kann man bequem vom Entwicklungs-PC aus auf den Gnublin zugreifen. (vgl. Embedded Projects

2013a)
Das Gnublin Board erfüllt das Anforderungsprofil vollständig.

IC-Board
Das IC-Board wird von der Firma: „IN-CIRCUIT“ gefertigt und ist als konfigurierbare Plattform konzipiert. Zur Analyse lag folgende Konfiguration vor: als Basis dient das Modell „ADB 4000“ (vgl. IN-CIRCUIT 2013) in der maximalen Ausbaustufe mit einem 8 Zoll Touchdisplay. Der mit 400 MHz getaktete ARM9-Prozessor sowie der 128MB starke DDR2 RAM befinden sich auf dem Small Outline Dual Inline Memory Module (SODIMM) mit der Bezeichnung „SAM9G45 SODIMM“ (vgl. IN-CIRCUIT 2014). Um das Board zu komplettieren, wird das gewählte SODIMM in das Entwicklerbord ADB 4000 gesteckt. Durch die Austauschbarkeit der Komponenten ist die Konfigurierbarkeit des IC-Boards ausgesprochen gut und kann an die speziellen Einsatzbedürfnisse angepasst werden. Der ADB-4000 verfügt über ein großes Potential an Schnittstellen, es werden hier ausschließlich die Schnittstellen mit Projektrelevanz aufgezählt Neben dem EthernetAnschluss verfügt es über vier USB-Ports. Ähnlich wie beim Gnublin-Board kann auf den ADB-4000 von einem anderem Rechner aus über ein Konsolen-Kabel zugegriffen werden. Bei dem ADB-4000 erfolgt der Zugriff allerdings über einen Universal Asynchronous Receiver Transmitter (UART)-Anschluss.
Das IC-Board erfüllt das Anforderungsprofil vollständig.

2.2.2 Software

Das Anforderungsprofil gibt hier ausschließlich die Nutzung des Modbus-Protokolls vor, da dies unabdingbar für die Kommunikation mit den Energiezählern ist. Die Auswahl der Software hängt bei diesem Projekt stark von der zuvor ausgewählten Hardware beziehungsweise Hardwarearchitektur ab. Je nach dem, welcher SOC verwendet wird, fällt die Wahl des Betriebssystems auf eines der folgenden.

Betriebssystem

Android
Android ist ein modernes Betriebssystem, das vor allem im „mobile device“-Bereich eingesetzt wird. Die Unterstützung für ARM-Prozessoren sowie die Integration von Touchdisplays und der anpassbare quelloffene Code unter Open Source machen Android immer attraktiver für die Bedürfnisse der embedded-Softwareentwicklung. Das SODIMM der Firma IN-CIRCUIT wird standardmäßig mit Android ausgeliefert.

Buildroot
Buildroot ist kein Betriebssystem, sondern ein Open Source-Werkzeug, das aus einer Ansammlung von Skripten besteht, die den Entwickler beim Erstellen des Linux-Filesystems unterstützen. Durch die hohe Konfigurierbarkeit lassen sich minimalisierte beziehungsweise stark optimierte Zielsysteme erzeugen. Die gewünschte Flexibilität sorgt allerdings auch für ein erhöhtes Maß an Komplexität, was sich unter anderem in dem mehrstufigen Entwicklungsprozess bis zum lauffähigen Betriebssystem widerspiegelt.
Die Toolchain, eine Ansammlung von Entwicklungswerkzeugen, um ausführbare Software für ein Zielsystem zu erzeugen, ist eine Grundvoraussetzung.
Für das IC-Board wird von IN-CIRCUIT eine angepasste und vorkonfigurierte Version von Buildroot angeboten. Die Konfiguration von Buildroot kann über das Terminalprogramm `menuconfig` angepasst werden. Durch den Aufruf des `make`-Befehls werden die Images für den Kernel, das Root Filesystem (RootFS) und den bootloader erzeugt. Nach dem erfolgreichen Erzeugen der Systemabbilder können diese über das Network File System (NFS)-Protokoll auf das Gerät übertragen werden.
Zusätzliche Software kann. je nachdem, ob ein Paketmanager beziehungsweise welcher Paketmanager installiert wurde, von diesem bezogen werden. Sollte kein Paketmanager installiert worden sein, so muss benötigte Software manuell aus dem Quellcode und dessen Abhängigkeiten kompiliert werden.

Gnublin Debian
Gnublin Debian basiert auf der Debian Version „Squeeze“ (Debian 6), das an die minimalistische Hardware des Gnublin's angepasst wurde. Dadurch kann man die Vorteile der Debian Distribution wie zum Beispiel des Package Manager nutzen und hat trotzdem einen geringen Speicher-Footprint. Des Weiteren sind zum einen die Treiber für alle verfügbaren Gnublin-Hardware-Module, wie zum Beispiel das ASCII-Display, und zum anderen die Gnublin-Application Programming Interface (API) integriert (vgl. Embedded Projects 2013b).
Die Installation von Gnublin Debian erfolgt über eine von „Embedded Projects“ bereitgestellte Software. Für die Installation wird von embedded projects jeweils ein Terminalprogramm sowie ein Programm mit grafischer Oberfläche bereitgestellt. Vor der eigentlichen Installation werden der gewünschten Bootloader, der Kernel und das RootFS auf der Herstellerseite ausgewählt und heruntergeladen. Mithilfe des Installationsprogramms in der grafischen Oberfläche werden die bezogenen Softwarekomponente ausgewählt und anschließend auf eine microSD-Karte übertragen.
Für die Installation über das Terminalprogramm muss dieses heruntergela-

den werden und mit dem Pfad zur microSD-Karte als `Argument` im Terminal gestartet werden. Die Programmroutine kümmert sich selbständig um den Bezug und die Installation der benötigten Dateien. Für die Verwendung wird eine funktionierende Python-Umgebung sowie eine Internetverbindung vorausgesetzt. (vgl. Embedded Projects 2013c).
Zusätzliche Software kann wie von Debian gewohnt über den Debian Package Manager beziehungsweise Advanced Packaging Tool (APT) installiert werden. Gnublin Debian verfügt von Haus aus nicht über eine grafische Benutzerschnittstelle, was für die Zielgruppe, die erfahrene Linux Anwender und Entwickler anspricht, aber kein Problem darstellen sollte.

Raspbian
Ähnlich wie Gnublin Debian ist Raspbian Debian eine angepasste Distribution auf der Basis von Debian Wheezy. Raspbian ist das offizielle Betriebssystem für den Raspberry Pi und wird von der Raspberry Pi Community entwickelt. In das Betriebssystem sind die Treiber für die Hardwareschnittstellen, wie zum Beispiel HDMI-Anschluss, bereits integriert.
Die Installation erfolgt über eine Netzwerk-Installation. Zuvor wird das Programm für die Netzwerk-Installation auf eine formatierte SD-Karte kopiert. Für die Installation ist eine Internetverbindung via Ethernet notwendig. Durch das Starten des Raspberry Pi mit der präparierten SD-Karte wird der Installationsdialog gestartet.
Zusätzliche Software kann wie von Debian gewohnt über den Debian Package Manager beziehungsweise APT installiert werden. Die Standart Raspbian Installation verfügt über eine grafische Benutzerschnittstelle.

Programmierschnittstellen - Application Programming Interface (API)

Eine API definiert die Schnittstelle zu einem Softwaresystem. Dieses Softwaresystem kann unter anderem ein selbständiges Programm oder eine Software-Bibliothek sein. Die Nutzung der Schnittstelle ermöglicht es dem Entwickler mit der Software, die die API bereitstellt, zu interagieren. Durch den Einsatz von Bibliotheken, auf die über ihre API zugegriffen wird, lassen sich wiederkehrende oder aufwendige Probleme, wie zum Beispiel der Zugriff auf eine Datenbank, lösen.

Gnublin-API
Von der Entwicklerfirma „Embedded Systems“ wird neben dem auf Debian für ARM-Prozessoren basierenden Betriebssystem auch eine API für den Gnublin angeboten und aktiv weiterentwickelt. Die Gnublin-API bietet

dem Entwickler einen einfachen Open Source-Zugang für die Steuerung der Hardwareschnittstellen. Der gut dokumentierte Funktionsumfang reicht von der Kommunikation zu diversen Gnublin-Hardware-Modulen sowie Peripherie über die Schnittstellen Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI) und GPIO bis hin zur Steuerung von Motoren. Zum aktuellen Zeitpunkt kann die Gnublin-API in Python- und C++Projekten eingesetzt werden

Modbus-Protokoll

Zu den Anforderungen des Projektes zählte, dass die Kommunikation zwischen dem SOC und dem Energiezähler mittels dem Modbus-Protokoll realisiert werden sollte. In diesem Kapitel sollen die Grundlagen des Protokolls erläutert werden.
Das Modbus-Protokoll wurde 1979 von der Firma Modicon (heute Schneider Electric) veröffentlicht. Im April 2004 wurde Modbus frei verfügbar und konnte ab diesem Zeitpunkt ohne Lizenzgebühren verwendet werden. Seit dem Jahr 2004 wird Modbus unter der Federführung von der „Modbus Organization“ und freiwilligen Entwicklern betreut und erweitert. Laut der Modbus Organization ist das Modbus-Protokoll ein „de facto Standard im industriellen Umfeld“. (vgl. Modbus Organization 2013).
Das Modbus-Protokoll ist in vier verschiedenen Version verfügbar: Modbus-Plus, Modbus-Remote Terminal Unit (RTU), Modbus-American Standard Code for Information Interchange (ASCII) und Modbus-Transmission Control Protocol (TCP).
Eine Übersicht darüber, wie die einzelnen Schichten des ISO/OSI-Models von den Modbus-Versionen RTU und ASCII genutzt werden gibt Tabelle 1.

Modbus-RTU und Modbus-ASCII

Modbus-RTU und Modbus-ASCII arbeiten nach dem Master/Slave Prinzip. Das bedeutet, dass ein Gerät (Master) ein oder mehrere Geräte (Slave) steuert. Auf das Projekt Visual Energy umgelegt bedeutet das, dass der SOC die Rolle des Masters übernimmt und die Werte der Energiezähler (Slave) abfragt. Der Unterschied in den Versionen RTU und ASCII liegt in der Codierung der Information. Während in der RTU-Version die Informationen binär übertragen werden, werden die Daten in der ASCII-Version als lesbare ASCII-Zeichen übertragen. Der Vorteil der Lesbarkeit in der ASCII-Version schränkt allerdings den Datendurchsatz deutlich ein.
Bei beiden Modbus-Versionen ist es möglich, dass ein Master mit bis zu 255 Slaves kommuniziert. (vgl. Schnell 2012, S. 208)

Schicht	**ISO/OSI**	**MB–RTU**	**MB–ASCII**
7	Application	MODBUS-Application Layer	
6	Presentation		
5	Session		
4	Transport		
3	Network		
2	Data Link	Master/Slave	
1	Physical	RS232/RS485	

Tabelle 1: Nutzung der ISO/OSI-Schichten durch Modbus(vgl. Schnell 2012, S. 310 - Tabelle wurde durch Verfasser angepasst)

Die Protocol Data Unit (PDU) ist bei beiden Modbus-Versionen dieselbe. Die PDU besteht aus den vier Feldern: Adresse, Funktionscode, Daten und Prüfsumme. Im Adressfeld ist hinterlegt, welches Gerät (Slave) angesprochen werden soll. Anschließend werden, je nach definiertem Funktionscode, auf dem Zielgerät die gewünschten Operation ausgeführt. Mögliche Operationen wären zum Beispiel das Lesen oder Schreiben von Registern. Die nötigen Informationen zum Durchführen der Operation sind im Datenfeld hinterlegt, mögliche Informationen wären zum Beispiel, welches Register verwendet werden soll. Bei der RTU-Version kommt ein 16 Bit cycle redundancy check (CRC) zum Einsatz. In der ASCII-Version wird wiederum der longitudinal redundancy check (LRC) verwendet. (vgl. Schwenninger 2003, S. 78 ff)

Modbus-RTU-Frame

Der Beginn eines neuen RTU-Frames wird durch eine Pause von größer gleich 3.5 Zeichen definiert. Die Übertragung des Frames darf nicht länger als 1.5 Zeichen unterbrochen werden, da sonst das Frame vom Empfänger verworfen wird. Die maximale Länge eines RTU-Frames beträgt 256 Bytes. Dies errechnet sich wie folgt: ein Byte für die Adresse, ein Byte für den Funktionscode, null bis 252 Bytes für den Datenteil und zwei Bytes für die CRC-Prüfsumme. (vgl. Schwenninger 2003, S. 79)

Modbus-ASCII-Frame

Wie schon erwähnt, wird in der ASCII-Version der Datenstrom für den Menschen lesbar übertragen, dieser Umstand fordert teilweise die Anpassung des ASCII-Frame. Ähnlich wie bei dem RTU-Frame ist der Beginn und das En-

de eines Telegramms definiert und erkennbar. Der ASCII-Frame besitzt die zusätzlichen Felder Start und Ende. Der Beginn wird durch einen „Kolon“ (:) und das Ende durch die Steuerzeichen carriage return (CR) und line feed (LF) kenntlich gemacht. (vgl. Schwenninger 2003, S. 80)

Libmodbus

Libmodbus ist eine in der Programmiersprache „C“ geschriebene Bibliothek die den Entwicklern den Umgang mit dem Modbus-Protokoll erleichtern soll. Anstatt eigene Modbus-Telegramme zu generieren, können Bibliotheksfunktionen verwendet werden, um die gewünschten Operationen, wie zum Beispiel das Auslesen von Registern, durchzuführen.
Da Libmodbus unter der GNU Lesser General Public License (LGPL) steht, ist die Bibliothek Open Source und somit frei verfügbar. Ein großer Vorteil liegt in der weitläufigen Portierung für unter anderem Linux, Mac OS X, FreeBSD und Win32. Die Installation unter Linux ist relativ trivial, da die meisten Distributionen die Bibliothek über die Paketquellen anbieten. Falls man nicht die Möglichkeit hat, Libmodbus über einen Paketmanager zu beziehen, so kann man die Bibliothek über das Makefile, das dem Quellcode beiliegt, kompilieren.

Programmiersprachen

Die Wahl der eingesetzten Programmiersprache kann erhebliche Auswirkung auf die Performance und die Auswahl an verwendbaren Ressourcen haben. Ein weiterer, nicht unerheblicher, Punkt ist die Ziel-Architektur sowie die Systemplattform, auf der die Software-Lösung im späteren Produktionsbetrieb ausgeführt werden soll. Essentielle Fragen an dieser Stelle sind zum Beispiel ob das Produkt plattformunabhängig sein muss, beziehungsweise welche Architekturen vom Programm unterstützt werden sollen.
Da die Programmiersprachen „C“ und „Java“ im Studiengang ausführlich behandelt wurden, möchte ich an dieser Stelle die Gelegenheit nutzen, die aufstrebende Programmiersprache „Python“ vorzustellen.

Python

Python ist eine interpretierte plattformunabhängige dynamische sowie stark typisierte Sprache. Die Version Python0.9 wurde 1991 durch Guido van Rossum veröffentlicht. Seit der Version Python2.1 wird Python von der Python Software Foundation (PSF) „Python Software Foundation“ verwaltet und steht als Open Source unter der Python Software Foundation License (PSFL) zur Verfügung. (vgl. Python Software Foundation 2013e; Python Software Foundation 2013d; Python Software Foundation 2013b)

Grundsätzlich wird von PSF empfohlen, bei neuen Projekten, die keine Abhängigkeiten zu bestehenden Python2-Implementierung haben, Python3 einzusetzen. (vgl. Python Software Foundation 2013c)
Wie eingangs schon erwähnt ist Python eine Interpretierte Sprache. Die Standard Implementierung von Python (CPython) wurde in der Programmiersprache „C“ umgesetzt. Neben CPython bestehen auch Implementierungen in „.NET“ (IronPython) sowie „Java“ (Jython). (vgl. Python Software Foundation 2013a)
Zu den grundlegenden Konzepten von Python zählt die gute Lesbarkeit des Quellcodes. Dies spiegelt sich beispielsweise in der Syntax wieder. Funktionsblöcke müssen durch Einrückungen kenntlich gemacht werden, in Java beispielsweise werden dafür „Akkoladen“ (geschwungene Klammern) („ “) verwendet. Der Python Interpreter berücksichtigt Funktionsblöcke ausschließlich, wenn diese eingerückt sind. Folgende Implementierung der Fakultätsfunktion demonstriert die Python3 Syntax.

```
def fakultaet(x):
        if x  1:
                return x * fakultaet ( x - 1 )
        else:
                return 1
```

An diesem Code-Beispiel fällt die für den Interpreter wichtige Formatierung auf, die wiederum die Lesbarkeit des Quelltextes steigert. Ein weiterer Beitrag zur Verständlichkeit des Quellcodes liegt in den simplen Kontrollstrukturen. Python nutzt für Kontrollstrukturen keine Klammern wie beispielsweise Java. Ein weiterer Vorteil von Python liegt in den sehr umfangreichen Standardbibliotheken, die in der Python-Terminologie Built-in Funktionen genannt werden. Built-in Funktionen müssen nicht importiert werden, der Zugriff darauf erfolgt durch die Verwendung von Schlüsselwörtern.
Ein Kritikpunkt an Python ist die Thematik Geschwindigkeit. Es ist bekannt, dass die Implementierung von Python, zum aktuellen Entwicklungsstand, nicht an die Performance von Programmiersprachen wie beispielsweise „C“ heran reicht. (vgl. The Computer Language Benchmarks Game 2014)
Diese Problematik lässt sich allerdings durch bestimmte Techniken etwas entschärfen. Je nachdem ob das ganze Programm oder ausschließlich ein definierter Bereich als zeitkritisch eingestuft wird, gibt es verschiedene Optimierungsansätze. Sollte das ganze Projekt als zeitkritisch eingestuft werden, so können Steigerungen durch den Einsatz der alternativen Implementierung „PyPy“ erzielt werden.Je nach durchgeführtem Test ist PyPy zwischen 0.16

bis 6.3 mal schneller als CPython. (vgl. pypy.org 2014)
Sollte nur ein gewisser Aufgabenbereich zeitkritisch sein, so besteht die geläufige Praxis darin, den betroffenen Teil in eine performantere Programmiersprache wie zum Beispiel „C“ auszulagern.
Es ist möglich Python-fremde-Programme in einem Python-Programm einzubinden. Für die Einbindung stehen die zwei Möglichkeiten: `subprocess` und `ctypes` zur Verfügung. Durch subprocess lassen sich neue Prozesse unter Python starten. Das ausführende Python-Programm kann mit den Ein- und Ausgabe-Kanälen des neuen Prozesses gekoppelt werden.
Eine weitere interessante Alternative ist ctypes. Durch ctypes hat der Entwickler die Möglichkeit auf dynamische Bibliotheken, die in „C“ entwickelt wurden, zuzugreifen. (vgl. Python Software Foundation 2013f)

Kapitel 3

Analyse

3.1 Auswahl der Technologien

Dieses Kapitel beschäftigt sich mit der Auswahl der Technologien, die für die Entwicklungsphase des Projektes Visual Energy eingesetzt werden. Die Auswahl der Technologien basiert auf dem Anforderungskatalog des Projektes und der darauf aufbauenden State of the Art-Analyse des letzten Kapitels. Die hier dokumentierte Auswahl ist ausschließlich für die Entwicklungsphase relevant. Es kann durchaus sein, dass es bis zum Erscheinen des fertigen Produktes zu diversen Änderungen in der Soft- und Hardware-Architektur kommen kann.

3.1.1 Eingesetzte Hardware

System On A Chip

Das Rückgrat des Projektes wird durch den SOC gebildet. Der gewählte SOC dient im Soft- sowie auch Hardware-Aspekt als Schnittstelle zum Energiezähler. Zum einem wird das Programm, das die Kommunikation zum Energiezähler steuert, auf dem SOC ausgeführt und zum anderem ist der SOC physikalisch über eine Netzwerkschnittstelle mit dem Energiezähler verbunden. Wie im vierten Kapitel erwähnt standen der Raspberry Pi, der Gnublin und das IC-Board zur Auswahl. Die Analyse hat ergeben, dass alle Geräte die Anforderungen erfüllen. Bis auf das IC-Board handelt es sich allerdings bei den Geräten um experimentelle Plattformen.
Trotz der geringen Hardwarekapazitäten hat sich schlussendlich, zumindest für die Dauer der Entwicklungsphase, das Gnublin-Board durchgesetzt.

Die Entscheidung wurde auf der Basis folgender Punkte getroffen:

- ausgezeichnete API
- sehr gute Dokumentation und Wiki für Entwickler
- hervorragender Support, Emails wurden schnell und hilfreich beantwortet sowie telefonische Erreichbarkeit zu Werkszeiten.

Energiezähler

Der Energiezähler ist direkt im Elektroinstallations-Verteiler montiert und misst die verschiedenen elektrischen Parameter. Diese Parameter werden durch das Programm über das Modbus-Protokoll ausgelesen.
Für die Energiezähler wurde keine State of the Art-Analyse durchgeführt, da hier im Vorfeld definiert wurde, welcher Hersteller und welche Modelle zum Einsatz kommen. Grundsätzlich sollte allerdings auch die Verwendung von anderen Energiezählern, sofern sie das Modbus-Protokoll unterstützen, nach diversen Anpassungen möglich sein.[1]
Im Projekt Visual Energy kommen von der Firma Saia-Burgess als dreiphasiger Zähler das Modell ALE3 und als einphasiger Zähler das Modell ALD1 zum Einsatz.

3.1.2 Eingesetzte Software

Betriebssystem

Durch die Wahl des Gnublin' s als SOC liegt der Einsatz von Gnublin Debian als Betriebssystem nahe. Alternativ besteht allerdings auch die Möglichkeit, eine andere Distribution zu verwenden oder gar ein eigenes System selbst zu entwickeln.
Anzumerken ist an dieser Stelle noch, dass das Gnublin Debian, wie im embedded Bereich geläufig ist, ohne grafische Oberfläche ausgeliefert wird, um die geringen Geräte-Ressourcen zu schonen.
Durch die Verwendung von Gnublin Debian besteht die Möglichkeit, bei diesem oder späteren Projekten, die Gnublin-API zu integrieren.

Kommunikation

In der ersten Version von Visual Energy besteht die Kommunikation ausschließlich zwischen dem SOC und den Energiezählern. Im Moment ist eine

[1] Mögliche Anpassungen wären die Adressen der verwendeten Register um die Energie Werte auszulesen, sowie eventuelle Konfiguration wie zum Beispiel die Slaveadresse oder die Baudrate anzupassen.

weitere Netzwerkkommunikation noch nicht vorgesehen, für spätere Veröffentlichungen aber durchaus denkbar und sinnvoll.
In der aktuellen Planung ist ausschließlich der lesende Zugriff auf die Energiezähler vorgesehen, bei Bedarf lässt sich dies aber auch auf den schreibenden Zugriff ausweiten.
Für die Kommunikation wird die in "C"programmierte Bibliothek Libmodbus Version3.0.5 eingesetzt. Diese Bibliothek unterstützt die Verwendung des Modbus-RTU Protokolls für das Projekt ausreichend.
In der aktuellen Version der Distribution Gnublin Debian ist Libmodbus nicht installiert oder über den Paketmanager erhältlich. Durch die Rücksprache mit Embedded Projects stellte sich heraus, das die Libmodbus Version3.0.5, aus dem Quellcode kompiliert, stabil läuft und bereits in der Community verwendet wurde.

Programmiersprachen

Als primäre Programmiersprache kommt die Programmiersprache Python in der dritten Version zum Einsatz. Neben der Möglichkeit der Objekt-Orientierten Softwareentwicklung bietet Python die Vorteile einer einfachen und leicht lesbaren Syntax. Durch die gute Lesbarkeit des Quelltextes soll eine bessere Wartbarkeit erzielt werden.
Ein weiterer Grund für die Entscheidung besteht in der guten Interoperabilität zwischen Python und anderen Programmiersprachen.[2] Dadurch ist die Möglichkeit gegeben, Visual Energy in zukünftigen Projekten beziehungsweise bestehenden Produkten zu integrieren.
Die Kommunikation mit den Energiezählern wird in die Programmiersprache "Cäusgelagert. Die Entscheidung der Auslagerung basiert auf der Annahme eine höhere Performance zu erzielen. Ein weiterer Grund besteht in der Verwendung der oben erwähnten Bibliothek Libmodbus, die in der Programmiersprache "Cëntwickelt wurde.
In der aktuellen Version der Distribution Gnublin Debian ist Python in der Version2 installiert, die benötigte Version3 lässt sich aber ohne Probleme aus dem Quellcode kompilieren.

[2] Ausführliche Informationen befinden sich im Abschnitt Python

3.2 Spezifikation

Die Auswahl der Technologien für das Projekt Visual Energy ist unter anderem das Ergebnis der State of the Art-Analyse des zweiten Zwischenbericht. Ein weiterer Aspekt der Auswahl kam durch die gegebenen Rahmenbedingungen des Projektes zustande. Auf der Basis der ausgewählten Technologien, die an dieser Stelle in tabellarischer Form jeweils für die Hardwarekomponenten (siehe Tabelle 2) sowie auch die Softwarekomponenten (siehe Tabelle 3) definiert wurden. Ausgehend von der Spezifikation wird die weitere Entwicklung durchgeführt.

System on a Chip:	Gnublin-LAN
Einphasiger Energiezähler:	Saia-Burgess ALD1D5FD00A2A00
Dreiphasiger Energiezähler:	Saia-Burgess ALE3D5FD10C2A00

Tabelle 2: Spezifikation der Hardwarekomponenten

Betriebssystem:	Gnublin Debian
Primäre Programmiersprache:	Python3.1
Sekundäre Programmiersprache:	C gcc
Kommunikationsprotokolle:	Modbus-RTU
Bibliotheken	Libmodbus3.0.5

Tabelle 3: Spezifikation der Softwarekomponenten

Kapitel 4

Entwicklung

In der Planungsphase werden allgemeine Überlegung zu der Umsetzung des Projektes getätigt, die anschließend bewertet und im Kapitel Implementierung realisiert werden. Da die Hardwarekomponenten definiert wurden behandeln das folgende Kapitel ausschließlich die Lösungsansätze beziehungsweise Dokumentationen der Softwarekomponenten.

4.1 Überlegungen zur Softwarearchitektur

Zum aktuellen Stand kann die Software in vier Aufgabenbereiche aufgeteilt werden: Steuerung, Kommunikation, Domäne und Persistenz. Das Steuerungs-Modul bildet zum einem den zentralen Einstiegspunkt in das Programm, anderem delegiert es Aufgaben an die entsprechenden Stellen. Für die Kommunikation mit den Energiezählern ist das Kommunikations-Modul verantwortlich. In der Domäne werden die zu überwachenden Energiezähler und ihre Messwerte verwaltet. Für die Speicherung der Daten dient das Persistenz-Modul.
Um einem gewissen Maß an Flexibilität zur gewährleisten sowie um Änderungen beziehungsweise spätere Erweiterungen zu ermöglichen, ist es von Vorteil wenn die einzelnen Schichten so weit wie möglich entkoppelt werden. Darunter versteht man, dass es für die einzelnen Schichten unerheblich ist, von welcher Quelle sie ihre Informationen und Befehlsdelegationen erhalten und jene somit leichter austauschbar sind.
Um die Wartbarkeit und Effizienz des Quellcodes zu maximieren sollte schon bei der Modellierung auf ein hohes Maß an Kohäsion geachtet werden. Unter Kohäsion versteht man, wie effektiv die einzelnen Strukturen ausschließlich ihren definierten Aufgabenbereich abdecken. Dies bedeutet anhand eines „bildlichen Beispiels“ demonstriert: die Struktur Personalbüro die Opera-

tionen „Person einstellen“ und „Person entlassen“ anbietet, nicht aber die Operation „Kundendaten anlegen“. Durch ein hohes Maß an Kohäsion wird der Code-Verdopplung vorgebeugt und der Anteil des wiederverwendbaren Quelltext maximiert.

4.1.1 Entwurf des Modells

Nachdem die einzelnen Schichten - auf der Basis ihrer Kompetenz-Bereiche - definiert wurden, liegt der nächste Schritt in der Überlegung wie die Interaktion der Schichten untereinander umgesetzt werden können. An dieser Stelle werden die verschiedenen Entwürfe, die im Entwicklungsprozess optimiert wurden, vorgestellt.

Erster Entwurf

Aktuell wird von dem zentralen Steuerungs-Modul aus eine neue Kommunikation initiiert. Zum besseren Verständnis des allgemeinem Kommunikationsablaufes dient das Flussdiagramm im Anhang A (siehe Abb.: 3).
Die Steuerung ruft das Modul für die Modbus-Kommunikation auf und übergibt die für die Kommunikation benötigten Daten. Das Kommunikationsmodul verwertet die erhaltenen Daten, um die Kommunikation zum gewünschten Energiezähler aufzubauen und sendet das erzeugte Modbus-Frame.
Grundsätzlich können während der Kommunikation zwei verschiedene Fälle eintreten. Der optimale Fall wäre es wenn die Kommunikation ohne Fehler abläuft. Dabei wurden die ausgelesenen Daten von dem Kommunikationsmodul an die Steuerung weitergegeben (*Case1*). Das alternative Szenario wäre es, das bei der Kommunikation ein Fehler aufgetreten ist (*Case2*). Sollte Case2 eingetreten entstehen zwei weitere Ablaufverhalten. Je nach Fehlerquelle kann entweder versucht werden, die Kommunikation erneut aufzubauen (*Case2.1*), oder die Kommunikation wird endgültig abgebrochen (*Case2.2*). Im Falle eines Abbruches wird ein spezifischer Fehlercode retourniert. Nach dem Erhalt der Daten leitet das Steuer-Modul die Informationen an das Kommunikations-Modul weiter, um sie schlussendlich in einer geeigneten Struktur abzulegen[1]

Aus diesem Ansatz wurde folgendes Schichtenmodell entworfen. (siehe Abb.: 1)

[1]Bei der aktuellen Planung werden nur Daten bei erfolgreicher Kommunikation gespeichert, dies entspricht ausschließlich *Case1* und *Case2.1*.

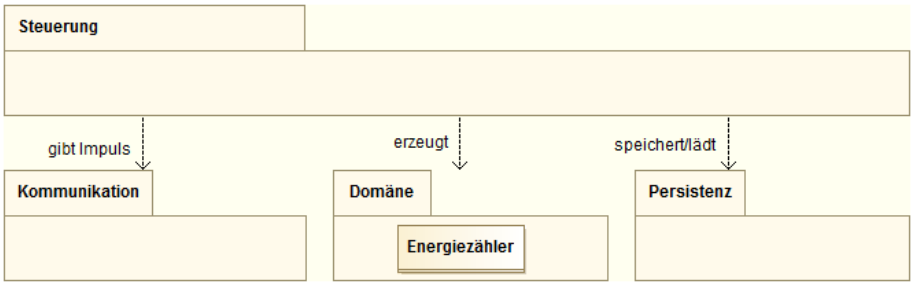

Abbildung 1: Schichtenmodell mit zentralen Ansatz

Optimierter Entwurf

Bei dem aktuellen Entwurf (siehe Abb.: 1) geht jede Interaktion von dem Steuerungs-Modul aus. Die einzelnen Module (untere Schicht) müssen für die Operationen allerding nicht das Steuerungs-Modul (obere Schicht) kennen, man spricht von einer einseitigen Abhängigkeit. Wenn die Module über wohldefinierte Schnittstellen verfügen, ist auch der - zuvor erwähnte - Austausch von Modulen möglich, wodurch eine Entkopplung der Schichten stattfindet. Durch die Entkopplung der Schichten wurde ein weiteres Konzept der Schichtenarchitektur - die Zugriffsvirtualisierung – realisiert. Der zentrale Ansatz muss nicht vollständig verworfen werden, allerdings sollten gewisse Stellen überarbeitet werden.
Wie zuvor erwähnt, bildet die Domäne die zu verwaltenden Energiezähler ab. Dies bedeutet, die gemessenen Werte gehören zu jeweils einem speziellen Zähler. Die hardwarespezifischen Information für den Auslesevorgang, wie zum Beispiel über welche Adresse oder welche Befehlssätze der Zähler verfügt, besitzt jeder Energiezähler selbst. Die Wissenskompetenz liegt dementsprechend bei dem Energiezähler beziehungsweise seiner Instanziierung aus der Domäne. Von dieser Annahme ausgehend, wäre es sinnvoll – sowie der Kohäsion dienlich – wenn ein Energiezähler (Instanziierung eines Domänenobjektes) selbst über einen eigenen geräte-spezifischen Befehlssatz und die entsprechenden Verbindungsinformationen verfügt. Aus diesen Änderungen würde folgendes neues Schichtenmodell entstehen (siehe Abb.: 2)

Die neue Version des Modells (siehe Abb.: 2) weist nun die Eigenschaften von einseitigen Abhängigkeiten zwischen den Schichten sowie einer realisierten Zugriffsvirtualisierung[2] auf. Das überarbeitete Modell kann dem Konzept der Schichtenarchitektur zugeordnet werden.

[2]Entkopplung zwischen den einzelnen Schichten

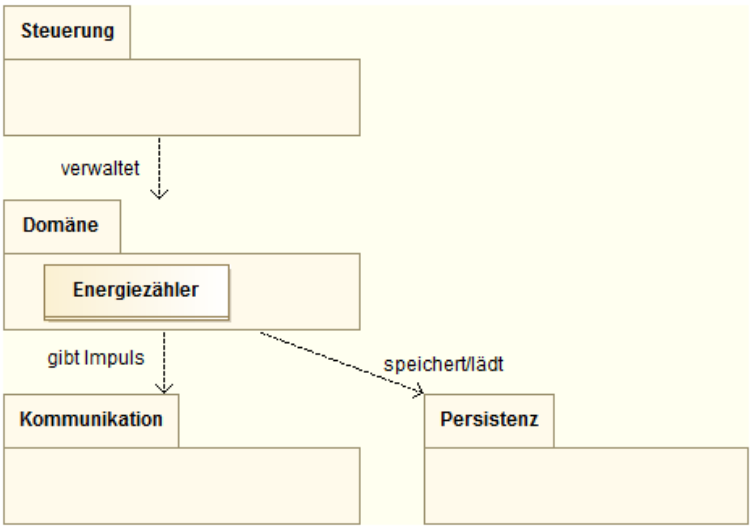

Abbildung 2: überarbeitetes Schichtenmodell

Kapitel 5

Implementierung

Nachdem im letzten Kapitel das Modell für die Softwarearchitektur definiert wurde (siehe Abb.: 2) kann nun - auf dieser aufbauend - die Implementierung geplant und umgesetzt werden. Neben den vorbereitenden Aufgaben wie dem Einrichten der Entwicklungsinfrastruktur wurde als Schwerpunkt in diesem Kapitel die Implementierung der Kommunikations-Schicht gewählt.

5.1 Aufbau der Entwicklungsinfrastruktur

Anhand der Spezifikation kann die Entwicklungsumgebung eingerichtet werden. Dazu muss unter anderem die Infrastruktur auf dem Gnublin installiert werden. Zum Aufsetzen der Infrastruktur zählt neben der Installation des Betriebssystems auch das Kompilieren der benötigten Software. Die Libmodbus-Bibliothek sowie Python3 sind nicht in den Paketquellen enthalten und müssen aus dem Quellcode kompiliert werden.
Das Hardware-Setup besteht aus dem SOC, der über einen USB-zu-RS485-Wandler mit den Energiezählern verbunden wird. (siehe Abb.: 4 im Anhang)

5.2 Allgemeines Vorgehen bei der Implementierung

Um die spätere Weiterentwicklung des Projektes zu erleichtern ist es wichtig, dass die implementierten Schritte zum einem dokumentiert und zum anderem gut verständlich sind.
Wie in der Spezifikation fixiert wurde, werden innerhalb des Projektes zwei Programmiersprachen zum Einsatz kommen. Bis auf die Funktionalität der

Modbus-Kommunikation wird die Implementierung in Python3.1 durchgeführt[1].
Das Kommunikations-Modul wird in „C“ implementiert.
Ein weiteres Ziel besteht darin, nach dem Konzept des Test-Driven Development (TDD) vorzugehen wodurch die Wartbarkeit und Softwarequalität gesteigert werden soll.
Ein elementares Kriterium des TDD basiert auf dem gezielten Einsatz von `Unit`-Tests. Ein wichtiger Aspekt bei der Entwicklung von `UNIT`-Tests besteht darin, dass der Test vor der Implementierung des Quellcodes geschrieben wird. Bei diesem Vorgehen muss sich der Entwickler im Vorfeld mit dem Funktionsumfang einzelner Strukturen und derer Schnittstellen auseinander setzen. Die einzelnen Module zählen erst dann als vollständig implementiert, wenn die `Unit`-Tests erfolgreich ausgeführt werden können. Dadurch werden schon in einem frühen Entwicklungsstadium Qualitätsmerkmale wie Beispielsweise wohldefinierte Schnittstellen und zielgerichtete Entwicklung erzielt.

Das Resultat der Risikoanalyse hat ergeben, dass die Implementierung der Kommunikation-Schicht am kritischsten eingestuft wird und deshalb als erstes umgesetzt werden sollte.

5.3 Implementierung der Kommunikations-Schicht

Die Aufgabe der Kommunikations-Schicht besteht darin, den anderen Schichten – in erster Linie der Domänen-Schicht – eine Schnittstelle für die Modbus-Kommunikation zur Verfügung zu stellen. Um dies zu realisieren müssen vorab die Voraussetzungen für die Kommunikation definiert sein.
Dies bedeutet zum einen abzuklären welche Informationen explizit für die Kommunikation benötigt werden und zum anderen festzustellen welche Optionen vonseiten des spezifischen Energiezähler-Modelles zur Verfügung gestellt werden.

[1] Es wird die Standard Implementierung CPythen verwendet.

5.3.1 Energiezähler spezifische Informationen

Für die Initialisierung der Kommunikation werden folgende Energiezähler spezifischen Daten benötigt:

- Baudrate
- Anzahl der Datenbits
- Anzahl des Stop-bit
- Art der Parität
- Adresse des auszulesenden Gerätes (Slave)

Die Baudrate gibt die Anzahl der übertragenen Symbole pro Zeiteinheit an und muss bei Sender und Empfänger gleich eingestellt sein. Die Anzahl der Datenbits gibt an wie viele Bits für die PDU verwendet werden. Das Stop-Bit gibt an ob ein oder zwei Stopbits verwendet werden. Es muss ausgewählt werden, ob das Paritäts-Bit gerade, ungerade oder nicht verwendet wird. Die Adresse des Slaves ist eine natürliche Zahl inklusive der Null, die angibt welches Zielgerät – das von dem Master verwaltet wird – angesprochen werden soll. Typenspezifische-Werte findet man in den Datenblättern der Energiezähler (siehe: *Datenblatt Modbus ALD1* 2011; und *Datenblatt Modbus ALE3* 2010).
Diese Daten bilden die Grundlage, um eine Verbindung aufzubauen. Dazu muss noch ergänzend angegeben werden auf welche/s Register zugegriffen wird. In diesen Registern werden spezielle Werte des Zielgerätes, wie beispielsweise die aktuelle Spannung, abgelegt. Da geplant ist, zwei verschiedene Versionen des Energiezählers einzusetzen, sollte an dieser Stelle die jeweiligen Registerbelegungen betrachtet werden. Während der einphasige Zähler 40 Register besitzt, werden beim dreiphasigen Zähler 52 Register bereitgestellt. Grundsätzlich sind die, bei beiden Zählern verwendeten Register, Register 1 – 40, mit den gleichen Funktionen belegt beziehungsweise bei den einphasigen Zähler unbelegt. Dieser Umstand basiert zum einem auf der Anzahl der unterstützten Phasen und zum anderem darauf, das der dreiphasige Zähler einen Tarife mehr verwendet. (vgl: *Datenblatt Modbus ALD1* 2011; und *Datenblatt Modbus ALE3* 2010)
Aus diesem Grund muss bei der Kommunikation zum einen sichergestellt sein, welches Modell angesprochen werden soll, zum anderen ob dieses Modell auch über die gewünschte Operation verfügt.

5.3.2 Planung des Modbus-Moduls

Für die eigentliche Modbus-Kommunikation via Modbus-RTU wird ein eigenständiges Programm (ComModbus) in der Programmiersprache „C“ geschrieben. Dieses Programm soll anschließend in das - in Python geschriebenen - Kommunikations-Modul integriert und von jenem verwendet werden.
Für die Integration, von in „C“ geschriebenen Programmen, gibt es in Python folgende zwei Ansätze:

- `subprocess`
- `ctypes`

Mit Hilfe von `subprocess` kann das Python-Programm zur Laufzeit neue (externe) Prozesse starten und mit den Ein- und Ausgabe-Kanälen des neuen Prozesses gekoppelt werden. Durch `ctypes` hat der Entwickler die Möglichkeit, zur Laufzeit auf dynamische Bibliotheken - die in „C“ entwickelt wurden - zuzugreifen.
Schlussendlich wurde ComModbus mithilfe von `subprocess` realisiert. Durch diese Entscheidung ist es möglich, ComModbus auch selbstständig in Form eines Konsolen-Programm oder als Modul in anderen Projekten einzusetzen.

5.3.3 Implementierung ComModbus

Für die Realisierung des TDD muss vor der Implementierung der Funktionsumfang und die Spezifikation für ComModbus definiert werden.

Spezifikation ComModbus

Das Programm ComModbus soll über den Terminal verwendet werden und benötigt keine grafische Oberfläche. Die auszuführenden Befehle sollen als Parameter beim Programm Aufruf übergeben werden. Die Reihenfolge der Parameter kann beliebig sein. Da die Energiezähler spezifischen Werte relativ statisch sind werden definierte Standard-Werte verwenden, solang diese nicht durch übergebene Parameter überschrieben werden. Die Ergebnisse sollen vollständig sein, bei dem auslesen von Doppelregistern,[2] sollen die Ergebnisse - vor der Ausgabe - zusammengeführt werden. Es soll immer nur ein Befehl - wie zum Beispiel lese aktuelle Spannung, ausgeführt werden. Der implementierte Befehlssatz soll erweiterbar sein. Die Notation der Parameter wird wie folgt definiert. Für Operationen werden Groß- und für Zähler spezifische Werte, Kleinbuchstaben verwendet.

[2]Datensätze die vom Speicherbedarf den eines Registers überschreiten können werden auf zwei Register aufgeteilt um möglichen Over-Flow-Errors vorzubeugen.

Befehlssatz ComModbus

Um den Befehlssatz für ComModbus definieren zu können muss zuerst betrachtet werden, welchen Funktionsumfang die verschiedenen Energiezählern zur Verfügung stellen. Durch die Verwendung von unterschiedlichen Modellen ist der Funktionsumfang teilweise - von Modell zu Modell - abweichend. In der folgenden Übersicht (siehe Tabelle 4) wird der Funktionsumfang der verschiedenen Zähler aufgeführt.[3]

Funktionsname	**Beschreibung**	**Wird von folgenden Modellen unterstützt**
Anzahl unterstützte Register	Je nach Modell 40 oder 52 Register	ALE3 und ALD1
Zähler T1 total	Energiezähler total Tarif 1	ALE3 und ALD1
Zähler T1 partial	Energiezähler partial Tarif 1	ALE3 und ALD1
Zähler T2 total	Energiezähler total Tarif 2	ALE3
Zähler T2 partial	Energiezähler partial Tarif 2	ALE3
URMS (*)	Wirkspannung der Phase	*
IRMS (*)	Wirkstrom der Phase	*
PRMS (*)	Effektive Wirkleistung der Phase	*
Cos phi (*)	Wirkfaktor der Phase	*

Tabelle 4: Verwendeter Funktionsumfang der Energiezähler ALE3 und ALD1 (vgl. *Datenblatt Modbus ALD1* 2011; und *Datenblatt Modbus ALE3* 2010)

Aus den verfügbaren Funktionen der Energiezähler wird der Befehlssatz für das Programm ComModbus hergeleitet. Der Aufruf der gewünschten Operation wird dem Programm durch die Übergabe des jeweiligen Großbuchstabens kenntlich gemacht. Der definierte Befehlssatz für das Programm ComModbus kann dem Anhang entnommen werden.

[3]Funktionen die mit einem Stern (*) versehen wurden sind in dreifacher Ausführung – für jede Phase separat – verfügbar.

5.3.4 Anwendungsfall ComModbus

Um den Programmablauf des Programmes ComModbus zu analysieren muss der Standard-Anwendungsfall definiert werden. In den folgenden Absätzen wird der Haupt-Anwendungsfall aufgezeigt, aus welchem der grobe Ablauf des Programmes ersichtlich wird. Die finale Version des Ablaufes ist in Form eines UML-Sequenzdiagrammes beigelegt (siehe Abb.: 5 im Anhang).

Akteure

Visual Energy und NutzerInnen

Vorbedingungen

Es wird davon ausgegangen, dass die benötigte Infrastruktur eingerichtet wurde und funktionstüchtig ist (siehe Aufbau der Entwicklungsinfrastruktur).

Nachbedingungen

ComModbus gibt den ausgelesenen und wenn benötigt berechneten Wert in der Konsole aus.

Standardablauf

1. Das Programm ComModbus wird mit der gewünschten Operation als Parameter aufgerufen.
2. Die übergebenen Parameter werden analysiert.
3. Zu dem gewünschten Energiezähler wird mithilfe der Parameter und der Standard-Konfiguration eine Verbindung aufgebaut.
4. Die verwendete Konfiguration wird ausgegeben.
5. Es wird geprüft ob das Zielgerät die Operation unterstützt.
6. Es wird der Wert aus dem benötigten Register gelesen.
7. Der ausgelesene Wert wird ausgegeben.

Alternativer Ablauf

1a Das Programm wird ohne Parameter gestartet.

1. Es werden verfügbare Optionen ausgeben.
2. Das Programm wird beendet.

2a Es wird eine nicht vorhandene Option als Parameter übergeben.

1. Es wird die verwendete Option – als Fehler markiert – ausgegeben.
2. Es werden verfügbare Optionen ausgeben.
3. Das Programm wird beendet.

2b Es wird eine Konfiguration als Parameter übergeben.

1. Die Standard-Konfiguration wird durch den/die übergebenen Parameter ersetzt.
2. Fortfahren bei Schritt3.

2c Es werden mehrere oder doppelte Operationen übergeben.

1. Sollte schon eine Operation von dem Programm registriert sein so werden die nächste/nächsten Operationen ignoriert.
2. Fortfahren bei Schritt3.

3a Die Verbindung kann nicht aufgebaut werden.

1. Analyse des Fehlercodes.
2. Ausgabe des Fehlergrundes mit spezifischen Lösungsvorschlag um die Fehlerursache zu beheben.
3. Das Programm wird beendet.

5a Das Zielgerät unterstützt die ausgewählte Operation nicht.

1. Es wird die verwendete Operation – als Fehler markiert – ausgegeben.
2. Es werden mögliche Operationen ausgeben.
3. Das Programm wird beendet.

6a Das Register kann nicht gelesen werden.

a. Startet wiederholten Leseversuch.
    1. Ausgabe der Fehlerbeschreibung.
    2. Fortfahren bei Schritt3.

b. Register kann trotz mehrmaligen Versuchen nicht gelesen werden.
   1. Ausgabe der Fehlerbeschreibung.
   2. Das Programm wird beendet.

6b Es wird auf ein Doppelregister zugegriffen.

1. Das Ergebnis wird auf der Basis der Werte - die von den einzelnen Registern ausgelesen wurden - berechnet.
2. Fortfahren bei Schritt7.

5.3.5 Realisierung von ComModbus

Der erste Schritt bei dem Implementierungsvorgang ist es, die übergebenen Parameter auszulesen. Durch die Verwendung der Funktion `getopt`, welche in der C-Bibliothek enthalten ist, kann das Auslesen der Übergabeparameter deutlich vereinfacht werden. Folgendes Beispiel bildet einen Aufruf mit Option und Parameter ab:

```
ComModbus -T 1
```

wobei `ComModbus` der Name des auszuführenden Programmes, `-T` die Option und `1` der Parameter ist. Solange es noch unbearbeitete Optionen gibt wird `getopt` in einer Schleife ausgeführt. Innerhalb dieser Schleife können die Optionen und Argumente in Variablen gespeichert und weiterverarbeitet werden.
Die Weiterverarbeitung wird in ComModbus durch einen `Switch-Case` abgehandelt. Jeder `Case` bildet die Business-Logik der jeweiligen Option ab. Durch dieses Vorgehen kann das Programm einfach erweitert werden. Bei Bedarf wird eine neue Option definiert, welche in einem weiterem `Case` realisiert wird.
Eine ausgewertete Option kann entweder eine Operation, eine Konfiguration oder eine ungültige Eingabe sein. Im Falle einer Konfiguration ersetzt diese die Standard Konfiguration.[4]
Wenn eine ungültige Operation übergeben wurde, wird der/die BenutzerInn informiert und das Programm beendet.
Bei einer gültigen Operation wird als erstes geprüft, ob das Programm bereits über eine aktuelle Operationsanweisung verfügt. Bei positiver Prüfung wird die neue Operation ignoriert.
Bei negativer Prüfung wird die neue Operation registriert. Diese Schleife wird so lange wiederholt, bis alle Optionen – die als Parameter beim Programmstart übergeben wurden – abgearbeitet sind.

[4]Siehe Abschnitt Konfigurationen des Befehlssatz ComModbus im Anhang

Anschließend wird überprüft, ob die registrierte Operation von dem gewünschten Gerät unterstützt wird. Sollte keine Unterstützung vorliegen, so wird der/die BenutzerInn informiert und das Programm beendet.
Sobald die Prüfung erfolgreich abgeschlossen ist, wird mit der Kommunikations-Routine fortgefahren.

Da vor der Modbus-Kommunikation nicht geprüft werden kann, ob der lesende Zugriff auf die benötigten Register möglich ist, wird an dieser Stelle eine Wiederholung implementiert.
Es kommt vor, dass für eine Operation mehrere Register – zum Beispiel Register 28, 29, 32 und 33 – benötigt werden. Wenn aus dieser Menge ein Register vorübergehend nicht gelesen werden kann, so muss das Programm mit einer Fehlermeldung abgebrochen werden.[5]
Der sicherste Weg der Vorbeugung besteht im Falle eines gesperrten Registers darin, den Lesevorgang zu wiederholen. Um eine Endlosschleife bei dieser Prozedur zu vermeiden, falls das Register längerfristig nicht verfügbar ist, wurden die Wiederholungen limitiert. Sollte zum wiederholten Male kein lesender Zugriff erfolgen, so wird das Programm mit einer Fehlermeldung beendet.

Während des Kommunikations-Ablaufes werden die einzelnen Schritte auf ihre Korrektheit hin überprüft. Sollte ein Fehler bei einem Zwischenschritt eingetreten sein so wird unverzüglich das `error handling` aufgerufen. Die vom `error handling` generierte spezifische Fehlermeldung wird vor dem Abbruch des Programmes ausgegeben. Sollte während einer aktiven Modbus-Verbindung ein Fehler auftreten, so wird vor dem Abbruch des Programmes die Verbindung geschlossen.

Sollte eine Operation ausgeführt werden, die mehrere Register benötigt, so muss vor der Ausgabe zuerst der Gesamtwert errechnet werden. Bei der Zusammenführung des Gesamtwertes werden entweder zwei oder vier Register[6] für die Operation verwendet. Bei dem Umgang mit Doppelregistern haben die einzelnen Register unterschiedliche Bedeutungen. Bei den verwendeten Energiezählern ist das Register mit der kleineren Adresse das höherwertigere Register. Jedes Register verfügt über einen 16bit vorzeichenlosen Speicherbereich und kann somit einen Zahlenbereich von bis zu 2^{16} (65.536) abbilden. (vgl. *Datenblatt Modbus ALD1* 2011; und *Datenblatt Modbus ALE3* 2010)

[5] Eine Ursache wäre der schreibende Zugriff des Zählers auf dieses Register.
[6] Was einem- oder zwei Doppelregistern entspricht

Register28	13
Register29	60383
Multiplikator	10^{-2} (0,01)
Einheit	kWh

Tabelle 5: Werte für die Doppelregisterberechnung

Beispiel:
Es soll der totale Wert des Tarifs1 ausgelesen werden. Dafür wird das Doppelregister mit der Adresse 28 und 29 benötigt.
Die weiteren Werte sind in Tabelle 5 angegeben.

Da das Register mit der niedrigen Speicheradresse das höherwertige Register ist, ergibt sich folgende Rechnung:

$$(Register28 \times Zahlenbereich + Register29) \times Multiplikator$$

Wenn man nun die spezifischen Werte aus Tabelle 5 einsetzt so erhält man folgende Gleichung:

$$(13 \times 65536 + 60.383) \times 0,01$$

Daraus ergibt sich, dass der totale Wert von Tarif1 momentan 9123,51 kWH beträgt.

5.3.6 Einbindung von ComModbus in der Kommunikationsschicht

Um ComModbus in Visual Energy einzubetten muss eine entsprechende Schnittstelle eingepflegt werden. Dafür wurde - nach dem Vorbild des Fassaden-Patterns - ein eigenes Python-Modul (`com_Modbus`) umgesetzt. Das Modul `com_Modbus` besitzt für jede verfügbare Operation des Programmes ComModbus eine eigene Python-Methode. Eine Aufgabe der Methoden besteht darin, die - beim Aufruf - erhaltenen Information zu verwerten und ComModbus via `subprocess` aufzurufen (siehe Abschnitt: Planung des Modbus-Moduls). Des weiteren sind die Methoden der Fassade auch dafür zuständig, dass die Input- und Output-Daten zwischen Python und „C“ kompatibel

sind. Durch diese Lösung kann innerhalb von Visual Energy bequem auf den gesamten Befehlssatz von ComModbus zugegriffen werden.

Um die Implementierung nach dem Schema des TDD zu realisieren werden vor der Umsetzung des Modules zuerst die Unit-Tests geschrieben. Da das Zusammenspiel zwischen der Kommunikations-Schicht und dem C-Programm ComModbus getestet werden soll, werden bei dem Test vordefinierte (hartkodierte) Werte für die Konfiguration verwendet. Getestet werden zum einen alle Operationen für den ALE3 sowie den ALD1. Des weiteren werden auch die Fälle getestet, bei denen die gewählte Operation nicht von dem jeweiligen Energiezähler unterstützt wird.
Nach dem Erstellen der Unit-Tests kann mit der Implementierung der Kommunikations-Schicht begonnen werden. Die Implementierung dieses Moduls zählt als abgeschlossen, wenn alle Unit-Tests erfolgreich durchlaufen wurden.

Kapitel 6

Evaluation

Das letzte Kapitel dieser Arbeit beschäftigt sich mit der Evaluation des Projektes Visual Energy. Die Evaluation ist in die Abschnitte **Ergebnisse** und **Ausblick** unterteilt. Im Abschnitt **Ergebnisse** befindet sich, neben der Zusammenfassung welche Aufgabenbereiche des Projektes abgeschlossen wurden, auch der Abschnitt **Tests & Benchmarks**. Eine Übersicht der potentiellen Entwicklungsmöglichkeiten des Projektes beziehungsweise dieses Themenbereiches befindet sich im letzten Abschnitt **Ausblick**.

6.1 Ergebnisse

Eine der Hauptanforderungen bestand darin, zwischen einem SOC und den - in der Spezifikation definierten - Energiezählern erfolgreich eine Verbindung aufzubauen und anschließend über diese die gemessenen Werte auszulesen. Das Auslesen der Daten konnte schon während der State of the Art-Analyse, mit Hilfe eines improvisierten C-Programmes und der Libmodbus-Bibliothek erfolgreich durchgeführt werden. Auf diesen Erkenntnissen und Technologien aufbauend sollte ein funktionelles und erweiterbares Programm mit folgendem Funktionsumfang entwickelt werden:

- Auslesen der Energiezähler
- Verwalten der Energiezähler
- Speichern der Messwerte

Das Auslesen der Energiezähler wurde - wie im Abschnitt Implementierung der Kommunikations-Schicht des Kapitel 5 beschrieben - vollständig implementiert und erfolgreich getestet.

Die Verwaltung der Energiezähler basiert auf einer - für Linux-Programme üblichen - Konfigurationsdatei. In dieser Datei werden die notwendigen Informationen für die Kommunikation gespeichert. Um eine geeignetere Basis für die spätere Weiterentwicklung zu gewährleisten wurde für die Energiezähler eine eigene Datenstruktur innerhalb des Python-Programmes implementiert. Diese Struktur enthält zum einen die Kommunikationsinformationen und zum anderem die verfügbaren Informationen des abgebildeten Energiezählers. Beide Versionen - Konfigurationsdatei und Datenstruktur - wurden vollständig implementiert und erfolgreich getestet.

Zum Zeitpunkt der Entwicklung wurde noch nicht definiert, in welcher Form die Messwerte (Snapshots) schlussendlich gespeichert werden. Um die implementierte Datenstruktur der Messwerte (siehe Abb.: 7 im Anhang) zu testen, wurde eine vorläufige Datenbank erstellt. Für diese Zwecke wurde die von Python mitgelieferte SQLite3 Datenbank mit einem provisorischen Schema (siehe Abb.: 9 im Anhang) verwendet.
Bei Tests wurden erfolgreich Snapshots der Zähler in die Datenbank geschrieben, allerdings ist die Persistenz-Schicht noch nicht ausgereift.

6.1.1 Tests & Benchmarks

Durch den Einsatz des TDD wurde kontinuierlich über den gesamten Entwicklungszeitraum hinweg getestet, wodurch keine kritische „big bang“-Testsituation zustande gekommen ist. Nachdem die Entwicklung eines Moduls - durch das erfolgreiche durchlaufen der Unit-Test - abgeschlossen wurde, konnte dieses Modul dem Integrationstest hinzugefügt werden. Bei dem Integrationstest wurde der Ablauf des Programmes auf die Kompatibilität der einzelne Module untereinander getestet.

Benchmarks wurden ausschließlich im Zuge einer Zeiterfassung durchgeführt. Dabei wurde der Standard-Versuchsaufbau verwendet. (siehe: Aufbau der Entwicklungsinfrastruktur des Kapitels 5) Der durchgeführte Test bestand aus der Abarbeitung folgender Punkte:

- Startup des Programmes
  - Starten und Ausführen des Python-Interpreters
  - Initialisierung des Energiezählers ALD1
  - Initialisierung des Energiezählers ALE3

- Daten lesen
  - Verbindungsaufbau zum Energiezähler ALD1
  - Snapshot für den Energiezähler ALD1 erstellen
  - Verbindungsaufbau zum Energiezähler ALE3
  - Snapshot für den Energiezähler ALE3 erstellen
- Snapshot speichern
  - Snapshot des Energiezähler ALD1 in der Datenbank ablegen
  - Snapshot des Energiezähler ALE3 in der Datenbank ablegen

Darstellung des Benchmarks

Die Ergebnisse des Benchmarks basieren auf 20 Testdurchläufen. Für den gesamten Test liegt der Mittelwert bei einer Dauer von 11,2 Sekunden. Dabei sind Messwerte in einem Bereich von 9,4 bis 15,5 Sekunden aufgetreten.

Davon entfallen auf den Start des Python-Interpreters zwischen 3.1 bis 4.5 Sekunden.

Interpretation des Benchmarks

Grundsätzlich sind die Ergebnisse des Benchmarks - auf Grund der geringen Anzahl an Messungen - nicht sehr aussagekräftig. Nichts desto trotz zeigen die Resultate folgende interessante Charakteristiken:

- starke Schwankungen in den Messwerten
- Subjektiv betrachtet eine lange Gesamtdauer

Allerdings muss an dieser Stelle angemerkt werden, dass die Tests auf einem SOC mit einem 180 MHz ARM9-Prozessor und 32 MB SDRAM durchgeführt wurden. Neben der spartanischen Hardware-Leistung kann ein weiterer Grund für die Performance darauf beruhen, dass es sich bei Python um eine interpretierte Sprache handelt und in diesem Punkt nicht mit maschinennahen Sprachen wie „C“ verglichen werden kann. Des weiteren sollte analysiert werden, ob eine Steigerung der Geschwindigkeit durch den Einsatz einer alternativen Implementierung wie „PyPy“ (vgl. pypy.org 2014) erzielt werden kann.

6.2 Ausblick

Da der Abschluss dieser Arbeit erst das Grundgerüst eines Systems mit großem Potential bietet, werden an dieser Stelle mögliche Ansatzpunkte für die weitere Entwicklung vorgestellt. Die hier vorgestellten Ideen lassen sich je nach Zielgruppe und Einsatzbereich des gewünschten Produktes optimieren, abändern oder erweitern.

dezentraler oder zentraler Ansatz

Eine wichtige Entscheidung besteht darin, ob für die zukünftige Weiterentwicklung ein zentraler oder dezentraler-Ansatz für das Projekt gewählt wird. Bei dem dezentralen Ansatz übernimmt der SOC ausschließlich die Funktionen, die Daten der Zähler auszulesen und an einen Server weiter zu leiten, welcher für weitere Funktionen und das Speichern der Daten zuständig ist. Sollte der Ansatz einer zentralen Lösung gewählt werden, so muss die aktuelle Arbeit um zwei neue Module erweitert werden. Zum einen sollte eine sinnvolle Persistenz- und zum anderen eine Visualisierungs-Lösung der Daten realisiert werden. Des weiteren sollten Überlegungen bezüglich einer Backup-Strategie angestellt werden.

Persistenz

Je nach gewähltem Ansatz und verwendeter Architektur können in diesem Bereich standardisierte Werkzeuge oder Speziallösungen eingesetzt werden. Sollte eine Lösung innerhalb einer ressourcen-schonendem Umgebung realisiert werden, wären möglicherweise SQLite oder RRDtool (siehe: Oetiker 2013) eine Option, welche genauerer betrachten sollte.

Visualisierung

Das Thema Visualisierung ist für das Produkt essentiell. Denn je ansprechender das Bild und die Strukturierung der gesammelten Werte, desto größer ist der Mehrwert für den/die BenutzerInnen. Eine „aufgeräumte“ Weboberfläche könnte denn einfachen Zugriff auf die Messwerte ermöglichen. Je nach Wunsch können die Graphiken auch mit Echtzeit-Animationen angereichert werden.

weitere Funktionalität

Derzeit ist das Projekt nur für das Messen und Aufzeichnen von Messwerten ausgelegt. Dies ist suboptimal, da bei der Thematik „Stromverbrauch“ deutlich mehr Potential verfügbar wäre. Neben einer informativen Darstellung wäre es denkbar, weitere Dienste anzubieten. Diese Dienste könnten beispielsweise als Webservice und/oder über ein Webinterface erreichbar und konfigurierbar sein. Szenarios wie Benachrichtigung bei Limit-Überschreitungen oder generierbare Abrechnung der konsumierten Einheiten pro Zähler beziehungsweise KundenInnen wären durchaus realisierbar.

6.3 Persönliche Meinung

Ich möchte diese Arbeit mit einer Reflexion über die gewonnen Erfahrungen aus meiner persönlichen Sicht abschließen.
Innerhalb dieses Projektes habe ich das erste mal Erfahrungen mit der Programmiersprache Python sammeln dürfen. Es hat mich immer beeindruckt, wie effektiv man mit dieser Programmiersprache entwickeln kann. Trotzdem sollten man sich bei neuen Projekten ernsthaft mit der Thematik, welche Python-Version zum Einsatz kommt, auseinandersetzen. Von offizieller Seite aus wird empfohlen, Python in der dritten Version einzusetzen, solange keine Abhängigkeiten zu Bibliotheken oder Paketen - welche in Python2 entwickelt wurden - vorhanden sind. (vgl. Python Software Foundation 2013c) Gerade dies ist allerdings nicht ganz einfach. Oft ließ sich auf den ersten Blick nicht evaluieren, in welcher Python-Version das gesuchte Framework vorliegt. Zum Zeitpunkt der State-of-the-Art Analyse im Sommer 2013 sind - subjektive betrachtet - deutlich mehr Frameworks und Pakete, die für meine Arbeit relevant gewesen wären, für Python2 verfügbar gewesen.

Eine weitere bereichernde Erfahrung war die Entwicklung nach dem Schema des TDD. Ein großer Vorteil für mich lag im Einsatz von Unit-Tests. Dadurch hatte ich zum einen die Sicherheit, dass die implementierten Module ihren definierten Aufgabenbereich korrekt ausführen. Zum anderen unterstützten die Test die Dokumentation des Projektes, denn bevor ein Unit-Test geschrieben werden kann, müssen die benötigten Parameter und Methoden definiert sein. Diese Definition lässt sich anschließend einfach in die Dokumentation überführen.
Allerdings muss erwähnt werden, dass TDD auch eine Herausforderung darstellen kann. Gerade in Situation in denen man „... nur mal schnell was ausprobieren will“ benötigt es doch ein gewisses Maß an Selbstbeherrschung,

um nicht in die Verhaltensmuster eines „schnellen Hacks“ zurückzufallen. Abschließend kann ich sagen, das ich das TDD weiter empfehlen kann und gerne wieder einsetze werde.

Abkürzungsverzeichnis

ALD1	Einphasige Energiezähler der Firma Saia-Burgess
ALE3	Dreiphasiger Energiezähler der Firma Saia-Burgess
API	Application Programming Interface
APT	Advanced Packaging Tool
ARM	Advanced RISC Machines
ASCII	American Standard Code for Information Interchange
CR	carriage return
CRC	cycle redundancy check
GPIO	General Purpose Input/Output
I2C	Inter-Integrated Circuit
ISO	International Organization for Standardization
LF	line feed
LGPL	GNU Lesser General Public License
LRC	longitudinal redundancy check
NFS	Network File System
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PSF	Python Software Foundation
PSFL	Python Software Foundation License

RFID	Radio-Frequency Identification
RISC	Reduced Instruction Set Computer
RootFS	Root Filesystem
RTU	Remote Terminal Unit
SDRAM	Synchronous Dynamic Random Access Memory
SOC	System On A Chip
SODIMM	Small Outline Dual Inline Memory Module
SPI	Serial Peripheral Interface
SSH	Secure Shell
TDD	Test-Driven Development
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver Transmitter
UMTS	Universal Mobile Telecommunications System

Literatur

Datenblatt Modbus ALD1 (2011). *Energiezähler mit integrierter serieller Modbus-Schnittstelle*. Saia-Burgess Controls Ltd.

Datenblatt Modbus ALE3 (2010). *Energiezähler mit integrierter serieller Modbus-Schnittstelle*. Saia-Burgess Controls Ltd.

Embedded Projects (2013a). *GNUBLIN-LAN - Wiki*. Embedded Projects. URL: http://en.gnublin.org/index.php/GNUBLIN-LAN (besucht am 03. 10. 2013).

— (2013b). *Gnublin Debian - Wiki*. Embedded Projects. URL: http://wiki.gnublin.org/index.php/Gnublin_Debian (besucht am 22. 08. 2013).

— (2013c). *Gnublin Debian Installer - Wiki*. Embedded Projects. URL: http://wiki.gnublin.org/index.php/Gnublin_Installer (besucht am 25. 08. 2013).

IN-CIRCUIT (2010). *Datenblatt ICnova ADB4000*. IN-CIRCUIT. URL: http://wiki.in-circuit.de/images/b/bd/305000045A_ICnova_ADB4000.pdf (besucht am 16. 01. 2014).

— (2013). *Spezifikation des ADB4000*. IN-CIRCUIT. URL: http://wiki.in-circuit.de/index.php5?title=ICnova_ADB4000 (besucht am 16. 01. 2014).

— (2014). *Spezifikation des SAM9G45 SODIMM*. IN-CIRCUIT. URL: http://shop.in-circuit.de/products/Home/ICnova-CPU-Module/2/ICnova-SAM9G45-SODIMM (besucht am 16. 01. 2014).

Modbus Organization (2013). *Modbus FAQ*. Modbus Organization. URL: http://www.modbus.org/faq.php (besucht am 16. 01. 2014).

Oetiker, Tobias (2013). *About RRDtool*. OETIKER+PARTNER AG. URL: http://oss.oetiker.ch/rrdtool/ (besucht am 22. 01. 2014).

Python Software Foundation (2013a). *About Python*. Python Software Foundation. URL: http://www.python.org/getit/ (besucht am 06. 10. 2013).

— (2013b). *General Python FAQ*. Python Software Foundation. URL: http://docs.python.org/3.3/faq/general.html#what-is-the-python-software-foundation (besucht am 04. 10. 2013).

Python Software Foundation (2013c). *Python2orPython3*. Python Software Foundation. URL: https://wiki.python.org/moin/Python2orPython3 (besucht am 04. 10. 2013).

— (2013d). *The History of Python*. Python Software Foundation. URL: http://python-history.blogspot.co.at/2009/01/brief-timeline-of-python.html (besucht am 04. 10. 2013).

— (2013e). *Why is Python a dynamic language and also a strongly typed language*. Python Software Foundation. URL: https://wiki.python.org/moin/Why%20is%20%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20%20language (besucht am 04. 10. 2013).

— (2013f). *ctypes — A foreign function library for Python*. Python Software Foundation. URL: http://docs.python.org/3.3/library/ctypes.html?%20highlight=ctypes#ctypes (besucht am 06. 10. 2013).

Raspberry Pi Foundation (2014a). *Raspberry Pi - About*. Raspberry Pi Foundation. URL: http://www.raspberrypi.org/about (besucht am 16. 01. 2014).

— (2014b). *Raspberry Pi - FAQ*. Raspberry Pi Foundation. URL: http://www.raspberrypi.org/faqs (besucht am 16. 01. 2014).

Schnell, Gerhard (2012). *Bussysteme in der Automatisierungs- und Prozesstechnik - Grundlagen, Systeme und Anwendungen*. 8. Aufl. Springer Viweg.

Schwenninger, Daniel (2003). "Implementierung von Modbus auf einer SPS und Anwendungsfall "Positionsregler für Siebdruckmaschine"". Diplomarbeit. Fachhochschule Vorarlberg.

The Computer Language Benchmarks Game (2014). *Python3 vs C gcc*. The Computer Language Benchmarks Game. URL: http://benchmarksgame.alioth.debian.org/u32/benchmark.php?test=all&lang=python3&lang2=gcc&data=u32 (besucht am 17. 01. 2014).

pypy.org (2014). *How fast is PyPy?* pypy.org. URL: http://speed.pypy.org/ (besucht am 17. 01. 2014).

Anhang A

Diagramme und Bilder

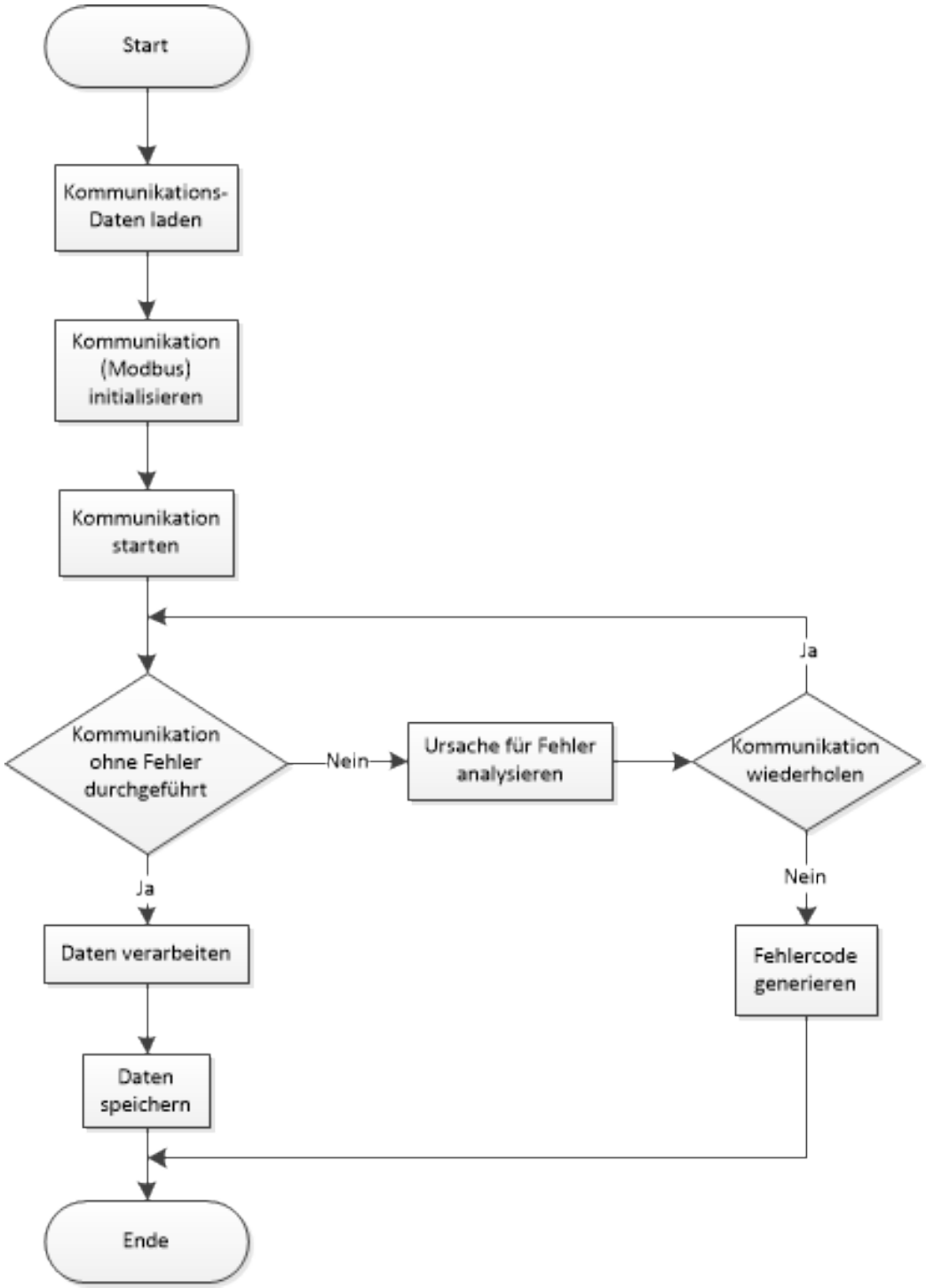

Abbildung 3: Flussdiagramm des Kommunikationsablaufes

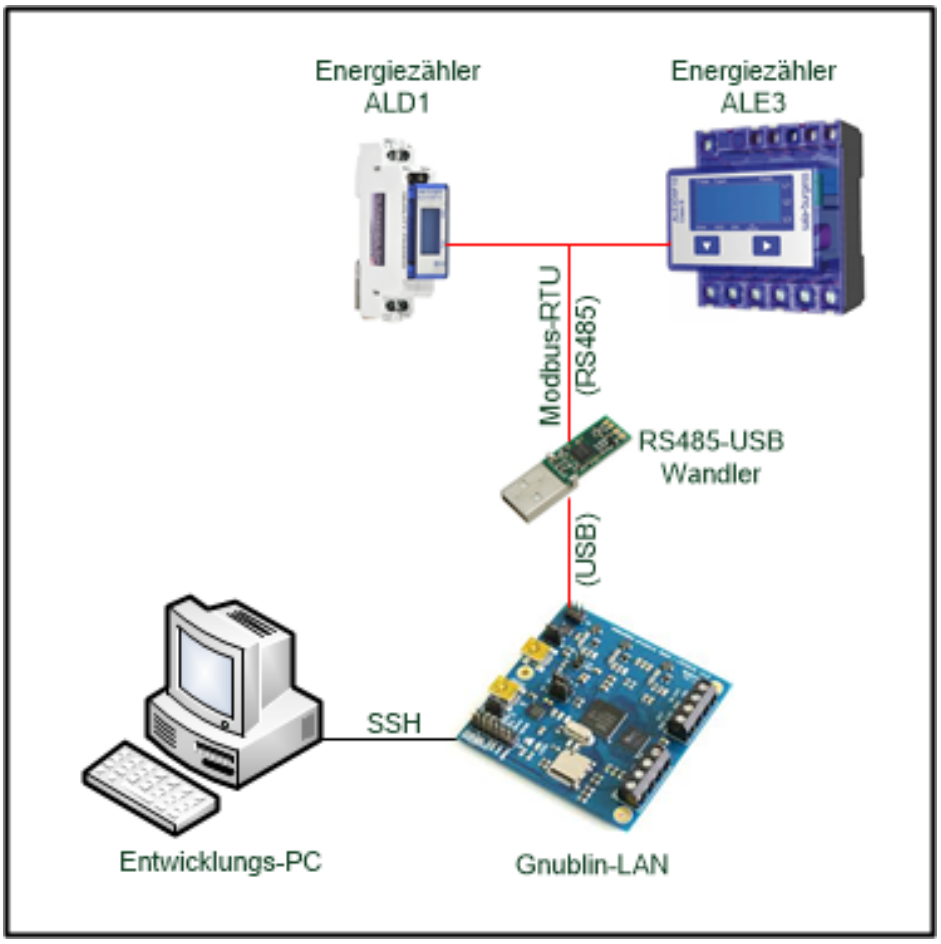

Abbildung 4: Hardware-Setup der Entwicklungsinfrastruktur

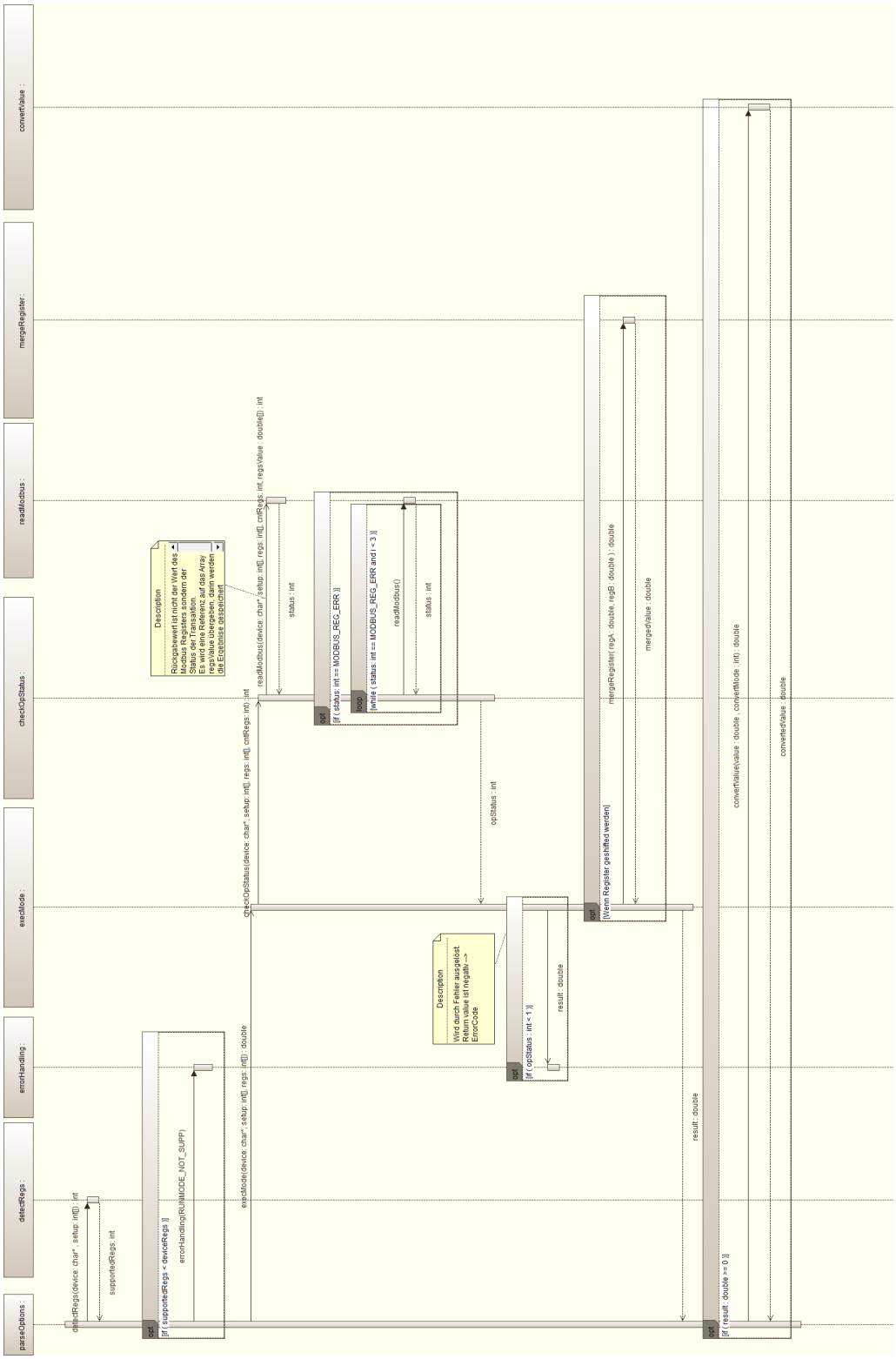

Abbildung 5: Sequenzdiagramm über den Ablauf des Programmes ComModbus

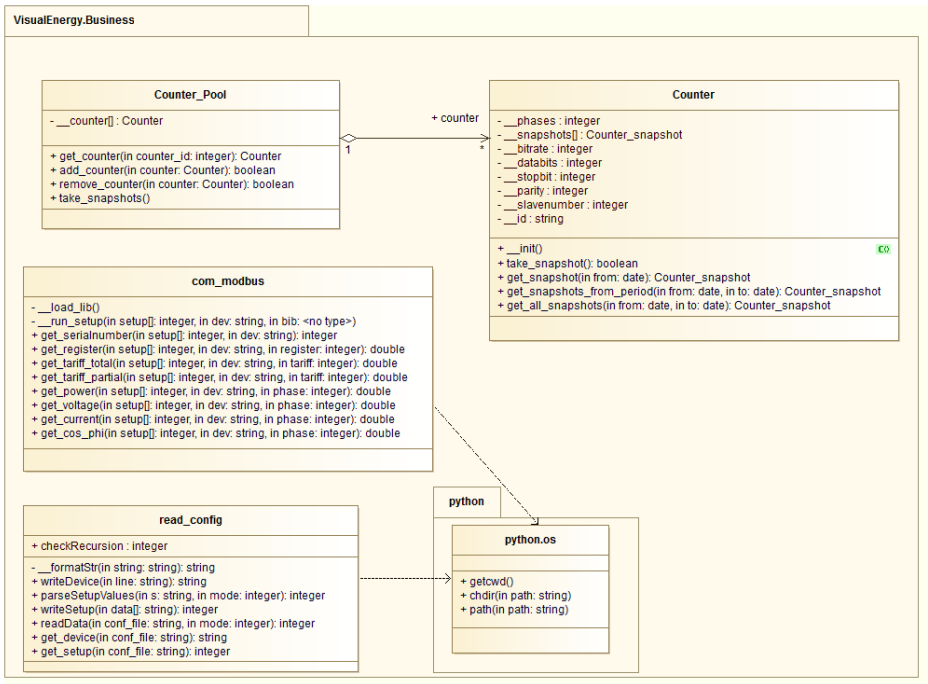

Abbildung 6: Klassendiagramm der Businessschicht

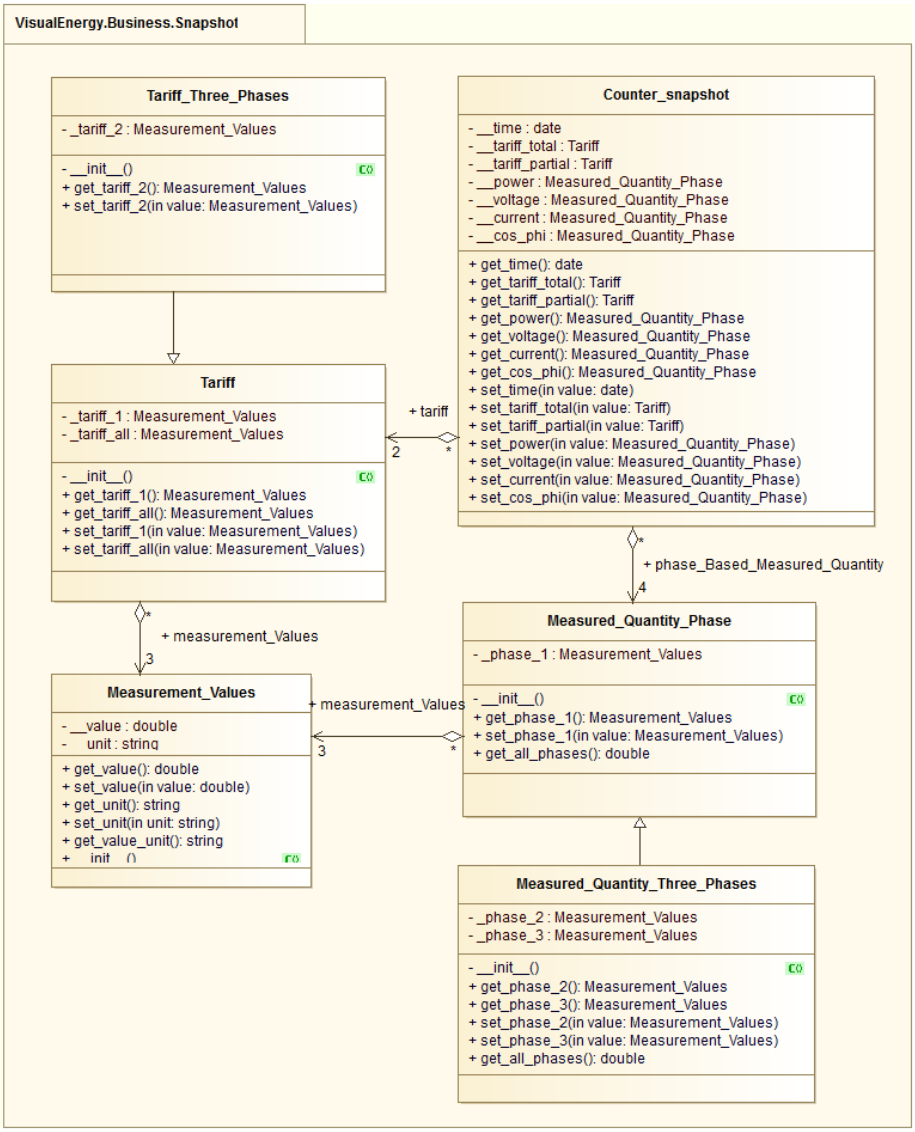

Abbildung 7: Klassendiagramm des Snapshot-Struktur

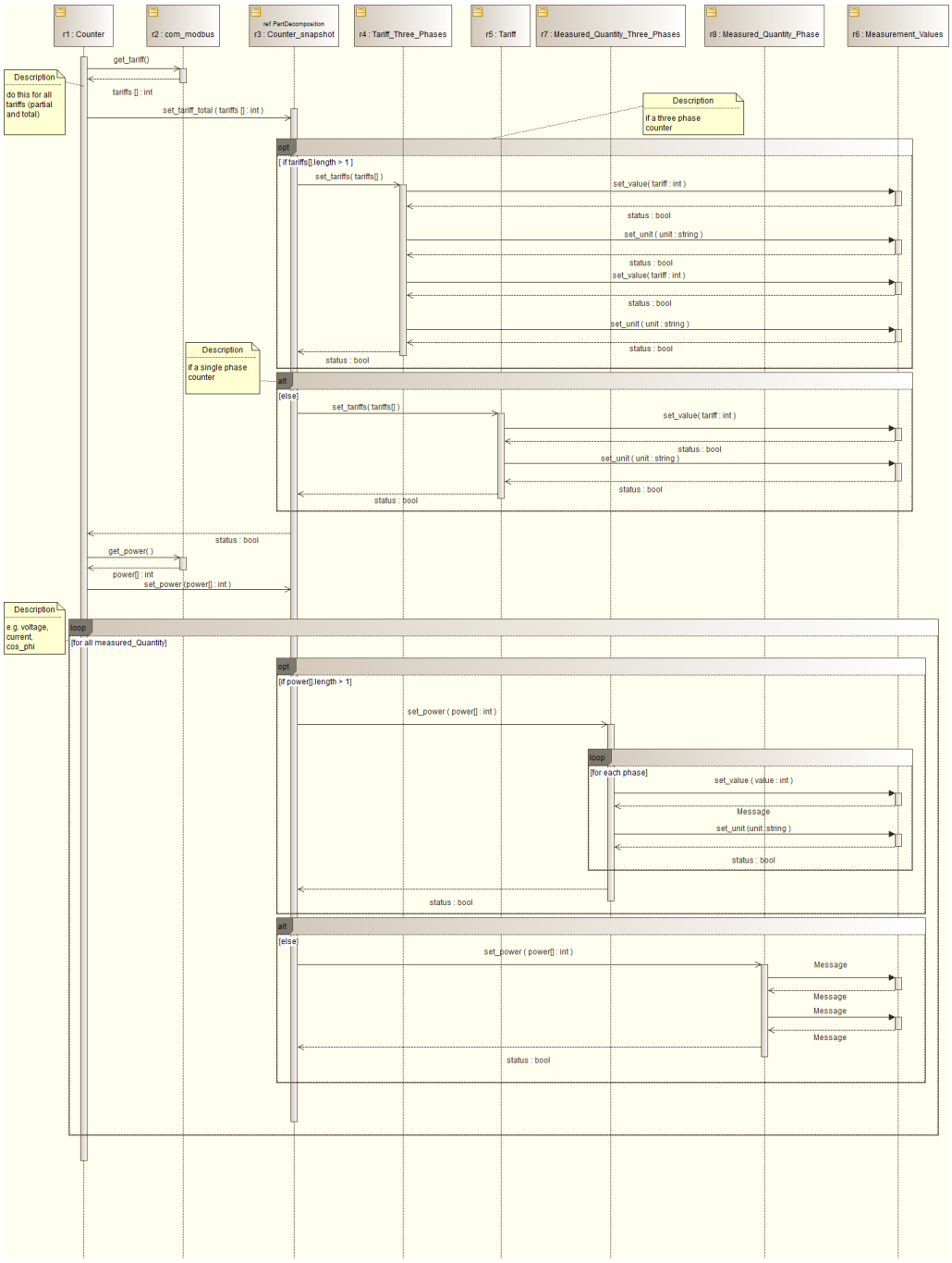

Abbildung 8: Sequenzdiagramm des Snapshot-Ablaufes

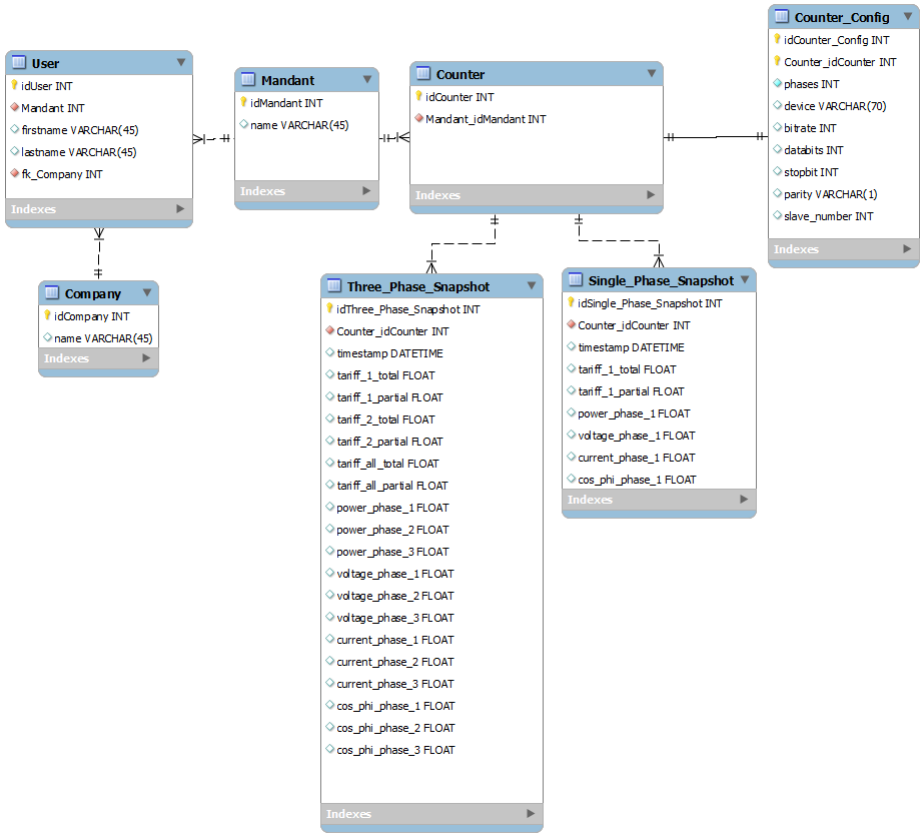

Abbildung 9: Aufbau des Datenbank-Modells

Anhang B

Befehlssatz ComModbus

Folgender Befehlssatz wird in der aktuellen Version von ComModbus unterstützt. Der Befehlssatz ist in Operation und Konfigurationen aufgeteilt und kann – bis auf die doppelt Belegung der Buchstaben – beliebig erweitert werden. Für die Verwendung der Buchstaben wurde folgende Notation eingeführt.

- Kleinbuchstaben für Konfigurationen
- Großbuchstaben für Operationen

Die Informationen werden als Parameter – in beliebiger Reihenfolge – an das Programm bei dessen Aufruf übergeben.

B.1 Konfigurationen

Die Konfigurationsdaten beschreiben die Energiezähler spezifischen Informationen die für die Modbus-Kommunikation notwendig sind. Wenn keine Konfigurationen als Parameter übergeben werden so werden die Standard Werte verwendet.

-i (Interface – Hardware-Schnittstelle)

```
ComModbus -i [Adresse der Hardware-Schnittstelle]
```

Mit dem Konfigurations-Befehl `-i [Adresse der Hardware-Schnittstelle]` kann dem Programm ComModbus mitgeteilt werden welcher Hardware Schnittstelle des Masters für die Modbus-Kommunikation verwendet werden soll.

Standard Wert: `ComModbus -i /dev/ttyUSB0`

-b (Baud-Rate)

`ComModbus -b [Baudrate]`

Mit dem Konfigurations-Befehl `-b [Baudrate]` kann dem Programm ComModbus mitgeteilt werden welche Baudrate verwendet werden soll. [Baudrate] muss eine natürliche Zahl sein. Die zu verwendende Baudrate kann dem Datenblatt des Energiezählers entnommen werden. Für eine fehlerfreie Kommunikation muss sichergestellt sein das sowohl Sender wie auch Empfänger mit der gleichen Baudrate ihre Informationen übertragen.

Standard Wert: `ComModbus -b 9600`

-p (Parität)

`ComModbus -p [E, O, N]`

Mit dem Konfigurations-Befehl `-p [E, O, N]` kann dem Programm ComModbus mitgeteilt werden welcher Art der Parität verwendet werden soll. Die zu verwendende Parität kann dem Datenblatt des Energiezählers entnommen werden.

Optionen:

- **E:** Paritäts-Bit ist gerade (**E**ven)
- **O:** Paritäts-Bit ist ungerade (**O**dd)
- **N:** Es wird kein Paritäts-Bit verwendet (**N**one)

Standard Wert: `ComModbus -p E`

-d (Daten-Bits)

`ComModbus -d [Anzahl Daten-Bits]`

Mit dem Konfigurations-Befehl `-d [Anzahl Daten-Bits]` kann dem Programm ComModbus mitgeteilt werden wie viele Datenbits verwendet werden soll. `[Daten-Bits]` muss eine natürliche Zahl sein. Die zu verwendende Anzahl der Daten-Bits kann dem Datenblatt des Energiezählers entnommen werden.

Standard Wert: `ComModbus -d 8`

-s (Stop-Bits)

```
ComModbus -s [Anzahl Stop-Bits]
```

Mit dem Konfigurations-Befehl `-s [Anzahl Stop-Bits]` kann dem Programm ComModbus mitgeteilt werden wie viele Stop-Bits verwendet werden soll. `[Stop-Bits]` muss entweder 1 oder 2 sein, wenn kein Paritäts-Bit verwendet wird muss 2 ausgewählt werden ansonsten 1. Die zu verwendende Anzahl der Stop-Bits kann dem Datenblatt des Energiezählers entnommen werden.

-n (slave number)

```
ComModbus -n [Adresse]
```

Mit dem Konfigurations-Befehl `-n [Adresse]` kann dem Programm ComModbus mitgeteilt werden welches Gerät (slave) verwendet werden soll. `[Adresse]` muss eine natürliche Zahl zwischen 1 und 247 sein[1]. Die zu verwendende Adresse muss am Gerät eingestellt oder von diesem abgelesen werden.

[1] Die Limitierung stammt von den Energiezählern nicht dem Modbus-Protokoll (siehe Datenblatt).

B.2 Operationen

-R (Register)

```
ComModbus -R [Adresse des Registers]
```

Die Adresse des Registers muss als positive Ganzzahl angegeben werden. Es wird der Wert des gewählten Registers ausgelesen. Die Registeradressen sind wie im Datenblatt angegeben zu verwenden. Es werden keine Validierungen des Ergebnisses durchgeführt. Diese Funktion dient in erster Linie Wartungsarbeiten.

Unterstützte Register

- Register 1 bis 40 (ALD1)
- Register 1 bis 52 (ALE3)

Die `[Adresse des Registers]` muss als positive Ganzzahl angegeben werden. Es wird der Wert des gewählten Registers ausgelesen. Die Registeradressen sind wie im Datenblatt angegeben zu verwenden. Es werden keine Validierungen des Ergebnisses durchgeführt. Diese Funktion dient in erster Linie für Wartungsarbeiten.

Beispiel

`ComModbus -R 2`
Liest das Register2 und gibt die Anzahl der unterstützten Register aus.

-T (Tarif – in kWh)

```
ComModbus { T [a, 1, 2, pa, p1, p2]
```

Mit dem Befehl T – Tarif können die Werte des Tarifes1 und -2 jeweils als partieller oder totaler Wert in kWh ausgegeben werden.

Optionen

a$^{\text{ALE3}}$:	Gibt die Summe aus Tarif1 (Total) und Tarif2 (Total) aus.
1:	Gibt den Wert von Tarif1 (Total) aus.
2$^{\text{ALE3}}$:	Gibt den Wert von Tarif2 (Total) aus.
pa$^{\text{ALE3}}$:	Gibt die Summe aus Tarif1 (Partiell) und Tarif2 (Partiell) aus.
p1:	Gibt den Wert von Tarif1 (Partiell) aus.
p2$^{\text{ALE3}}$:	Gibt den Wert von Tarif2 (Total) aus.

ALE3: Operationen werden ausschließlich von dem dreiphasigen Zähler unterstützt.

Beispiel

```
ComModbus -T p1
```
Gibt den partiellen Wert des Tarifs1 in kWh aus.

-P (PRMS – in kW)

```
ComModbus -P [a, 1, 2, 3]
```

Mit dem Befehl P – PRMS wird die effektive Wirkleistung auf der jeweiligen Phase beziehungsweise allen drei Phasen in kW ausgegeben.

Optionen

a$^{\text{ALE3}}$:	Gibt die Summe aus Phase 1 – 3 aus.
1:	Gibt denn Wert von Phase1 aus.
2$^{\text{ALE3}}$:	Gibt denn Wert von Phase2 aus.
3$^{\text{ALE3}}$:	Gibt denn Wert von Phase3 aus.

ALE3: Operationen werden ausschließlich von dem dreiphasigen Zähler unterstützt.

Beispiel

`ComModbus -P 1`
Gibt den Wert der effektiven Wirkleistung auf der Phase1 in kW aus.

-U (URMS – in V)

`ComModbus -U [1, 2, 3]`

Mit dem Befehl U – URMS wird die Wirkspannung auf der jeweiligen Phase in Volt ausgegeben.

Optionen

1:	Gibt denn Wert von Phase1 aus.
2$^{\text{ALE3}}$:	Gibt denn Wert von Phase2 aus.
3$^{\text{ALE3}}$:	Gibt denn Wert von Phase3 aus.

ALE3: Operationen werden ausschließlich von dem dreiphasigen Zähler unterstützt.

Beispiel

`ComModbus -U 1`
Gibt den Wert der Wirkspannung auf der Phase1 in Volt aus.

-I (IRMS – in A)

`ComModbus -I [a, 1, 2, 3]`

Mit dem Befehl I – IRMS wird des Wirkstromes auf der jeweiligen Phase beziehungsweise allen drei Phasen in Ampere ausgegeben.

Optionen

a[ALE3]:	Gibt die Summe aus Phase 1 – 3 aus.
1:	Gibt denn Wert von Phase1 aus.
2[ALE3]:	Gibt denn Wert von Phase2 aus.
3[ALE3]:	Gibt denn Wert von Phase3 aus.

[ALE3]: Operationen werden ausschließlich von dem dreiphasigen Zähler unterstützt.

Beispiel

```
ComModbus -I 1
```
Gibt den Wert des Wirkstromes auf der Phase1 in Ampere aus.

-C (Cos phi)

```
ComModbus -C [1, 2, 3]
```

Mit dem Befehl C – cos phi wird der Wirkfaktor auf der jeweiligen Phase ausgegeben.

Optionen

1:	Gibt denn Wert von Phase1 aus.
2[ALE3]:	Gibt denn Wert von Phase2 aus.
3[ALE3]:	Gibt denn Wert von Phase3 aus.

[ALE3]: Operationen werden ausschließlich von dem dreiphasigen Zähler unterstützt.

Beispiel

```
ComModbus -C 1
```
Gibt den Wert des Wirkfaktors auf der Phase1 aus.