Entwicklung einer Schnittstelle für die Anbindung von eingebetteten Systemen mittels REST an ein PHP basiertes Warenwirtschaftssystem

Masterarbeit

Erstprüfer:	Prof. Dr. Hubert Högl
Zweitprüfer:	Prof. Dr. Nik Klever

vorgelegt von:	Benedikt Sauter
	Matrikelnummer 915989
	Kettengässchen 6
	86152 Augsburg
	sauter@embedded-projects.net

Abgabetermin:	18.06.2010

Danksagung

Besonders bedanken möchte ich mich bei Prof. Dr. Hubert Högl, welcher mich während meines Studiums mit der Thematik Embedded-Systeme begeistert und redlich Unterstützt hat.

Erklärung

Hiermit versichere ich, die Arbeit eigenständig und nur unter Verwendung der angegebenen Hilfsmittel durchgeführt zu haben.

Augsburg 14. Juni 2010

Inhaltsverzeichnis

Einleitung

Warenwirtschaftssysteme sind heutzutage ein wichtiger Bestandteil jeder modernen Firma. Sie bilden das Rückgrat des Managements bzw. der Verwaltung und Steuerung der Geschäftsprozesse. Verschiedene Wissenschaften wie z.B. Wirtschaftsinformatik, Betriebswirtschaftslehre, Maschinenbau, Mathematik u.A. beschäftigen sich bereits seit Jahrzehnten mit der Entwicklung und Weiterentwicklung von Software zu dieser Thematik.

Im Zeitalter der Miniaturisierung des Computers und der Mikroprozessoren finden sich in Firmen immer mehr Embedded Systeme – in Form von einfachen Barcode-Scannern, Waagen bis hin zu modernen Kameras und tragbaren Funk-Computern – um Geschäftsprozesse und -abläufe besser steuern und regeln zu können.

Betrachtet man die Strukturen von Warenwirtschaftssystemen samt deren Prozesssteuerungen, so stellt man fest, dass sie den modernen Industrieautomatisierungen sehr ähnlich sind. In großen Firmen, die ganze Hallen mit Verpackungsmaschinen, Förderanlagen, automatisierten Prozessschritten etc. betreiben, ist diese Technik der Vernetzung und Ansteuerung aus der Großindustrie schon lange „Status Quo“. Die Netzwerke sind optimiert auf eine flexible Anbindung, dynamische Aufbauten, sichere Datenübertragung und ein z.T. integriertes Power-Management.

Wagt man einen Blick in kleine und mittelständische Unternehmen, bietet sich auf den ersten Blick ein ähnliches Bild. Beim näheren Betrachten bemerkt man jedoch, dass Hardware wie z. B. Barcode-Scanner oder externe Waagen meist über Umwege mit dem Warenwirtschaftssystemen verbunden sind. Außerdem fungieren Barcode-Scanner oftmals nur als Tastatur und geben einen eingescannten Wert direkt an der aktuell gesetzten Cursorposition aus. Auch die Software von digitalen Waagen legt die Werte nur regelmäßig in ein Netzwerklaufwerk ab, welches durch einen entfernten Prozess mittels Polling beobachtet wird.

Zu den eben genannten Umwegen und umständlichen Methoden kommt noch hinzu, dass abhängig von Branche und Firma der Ausbau einer optimalen Struktur zur Steuerung von Geschäftsprozessen mit Hardwareanbindung verschiedenste und individuelle Abläufe benötigt. Mittels Standardsoftware ist es nahezu unmöglich einen gemeinsamen Nenner zu finden. Auf Grund dessen existiert in diesem Bereich eine große Anzahl kleiner selbst entwickelter Anwendungen, die über verschiedenste Export- und Importschnittstellen Daten von Prozessen an Warenwirtschaftssysteme und beteiligte Software übergeben. Daraus

ergibt sich häufig ein Sammelsurium unterschiedlichster autonomer Softwarebereiche, die umständlich miteinander verknüpft werden müssen, um ein optimales und individuell angepasstes Ergebnis zu erzielen.

Im Rahmen der Masterarbeit wird für genau dieses Problem eine standardisierte Schnittstelle zur Anbindung externer Peripherie an kleine und mittelgroße Warenwirtschaftssysteme entwickelt, welche sich leicht und schnell ohne viel Aufwand auch in bestehende Strukturen integrieren lässt. Die Schnittstelle umfasst folgende Punkte: Anbindung der Geräte mittels einer Adapter-Box, Vernetzung der Komponenten mittels TCP/IP basierend auf Ethernet, eine Webschnittstelle als REST Implementierung, eine Bibliothek für Softwarebetreiber von Warenwirtschaftssystemen, eine zentrale Management-Konsole für die Verwaltung der Komponenten und als wesentlicher Bestandteil des Konzepts ein durchgängiges intelligentes und integriertes Power-Managment (Zeit- und Ereignisgesteuert) für alle Teilnehmer des Netzwerks, um Strom und Kosten zu sparen.

Eine Installation des Netzwerks wurde im Rahmen der Arbeit bei der Firma embedded projects GmbH in Augsburg aufgebaut. Die Anwendung wird dort seit März 2010 eingesetzt und fortwährend weiterentwickelt.

1. Stand der Technik / Anforderungen

Eine Analyse des Stands der Technik und die Betrachtung der Anforderungen an solch eine Lösung bedarf eine globale Sicht auf die verschiedenen Bereiche, die sich auf den Inhalt der Arbeit auswirken: Warenwirtschaftssysteme, Datenkommunikation, Internetprogrammierung und die Entwicklung von Embedded Systemen.

Das Warenwirtschaftssystem ist die betriebswirtschaftliche Grundlage. Themen wie Geschäftsprozesse, Lager- und Distributionssysteme sowie Warehouse-Management bilden den Kern dieses Bereiches.

Die Datenkommunikation ist die Brücke zwischen der Adapter-Box dem Embedded-System und der Internetprogrammierung bzw. den betriebswirtschaftlichen Anforderung des Netzwerks. Qualitäts- und Dienstgüteanforderungen beeinflussen die Wahl der Schnittstellen, Übertragungsmedien und Programmierung des Netzwerkverkehrs.

Die Internetprogrammierung ist das späterer Werkzeug, von dem das Netzwerk aus betrieben und angesteuert wird. Der aktueller Stand der Technik und deren Trend der Entwicklungen auf dem Markt von Programmiersprachen und Technologien im Bereich Internet muss betrachtet werden.

Das Embedded-System ist der Bereich für die Entwicklung einer optimalen Adapter-Box (elektronische Schaltung) zur Ansteuerung der Geräte. Für den Betrieb des Netzwerkes müssen die Bauteile und Schaltungen für die Dimensionierung des Embedded-Systems entsprechend gewählt werden.

Die Bereiche Warenwirtschaftssysteme, Datenkommunikation und Internetprogrammierung werden nachfolgend dargestellt, der Bereich Embedded-Systeme bildet den Hauptteil der Masterarbeit ab Kapitel 4.

1.1. Warenwirtschaftssysteme

Um die Anforderungen im weiteren Verlauf der Arbeit besser darstellen zu können, werden zunächst einige Begriffe aus dem Zusammenhang mit Warenwirtschaftssystemen erklärt:

Ein **Warenwirtschaftssystem** (abgekürzt *WWS* oder *WaWi*) ist ein Modell zur Abbildung der Warenströme im Geschäftsprozess eines Unternehmens. Der Begriff wird überwiegend im Zusammenhang mit Software für Disposition und Logistik verwendet [15].

Was ist ein Geschäftsprozess und was ist ein Lager- bzw. Distributionssystem?

Ein **Geschäftsprozess** beschreibt typischerweise eine Folge von Schritten welche den gesamten Prozess ergeben. Ein Geschäftsprozess kann wiederum Teil eines weiteren Geschäftsprozesses sein [1]. Typische Geschäftsprozesse sind Verkauf einer Ware, Bestellung von Material für das Lager, Produktion einer Ware A für einen Kunden B.

Lager- und Distributionssysteme sind auf ein spezielles Anforderungsprofil zurecht geschnittene Lösungen, die sich entsprechend der Vielfalt der technischen Prozesse und organisatorischen Gestaltungsmerkmale mehr oder weniger stark unterscheiden [2].

Die Masterarbeit verbindet die Strukturen des Warenwirtschaftssystems für die Geschäftsprozesse mit der Peripherie für Lager- und Distributionssysteme (siehe Abbildung 1). Zum Beispiel wird beim Verkauf einer Ware (Geschäftsprozess) das Gewicht mit Hilfe einer digitalen Waage (Peripherie) erfasst und die jeweilige Artikelmenge mit dem Barcodescanner (Peripherie) aus dem Lager gebucht (Lager- und Distributionssysteme).

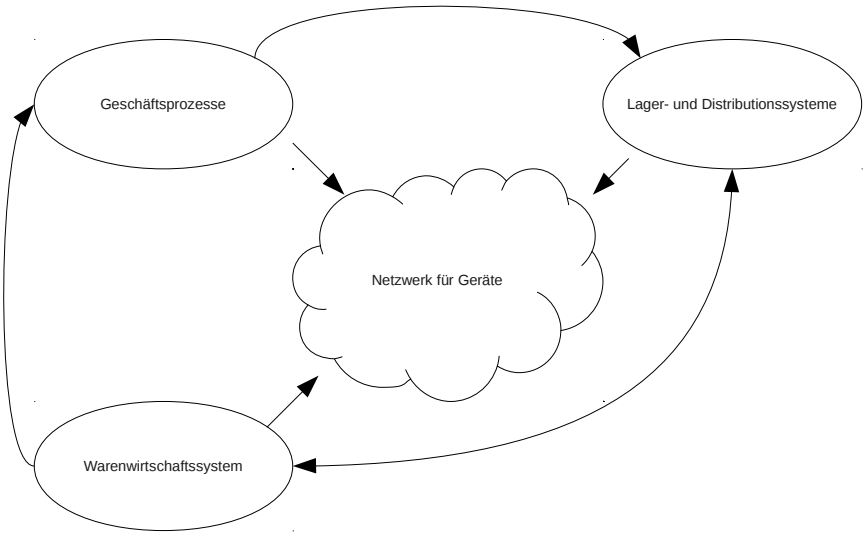


Abbildung 1: Integration der Netzwerkschnittstelle

Typische Anforderungen und Zusammenhänge für dieses Zusammenspiel sind in dem Organigramm in Abbildung 2 dargestellt. Es zeigt den internen Aufbau der Testinstallation der Firma embedded projects GmbH, ein kleines Unternehmen mit ca. drei bis vier Mitarbeitern, das mitunter einen Online-Shop betreibt. Die Firma ist ein typischer Anwender für das Netzwerk der Masterarbeit.

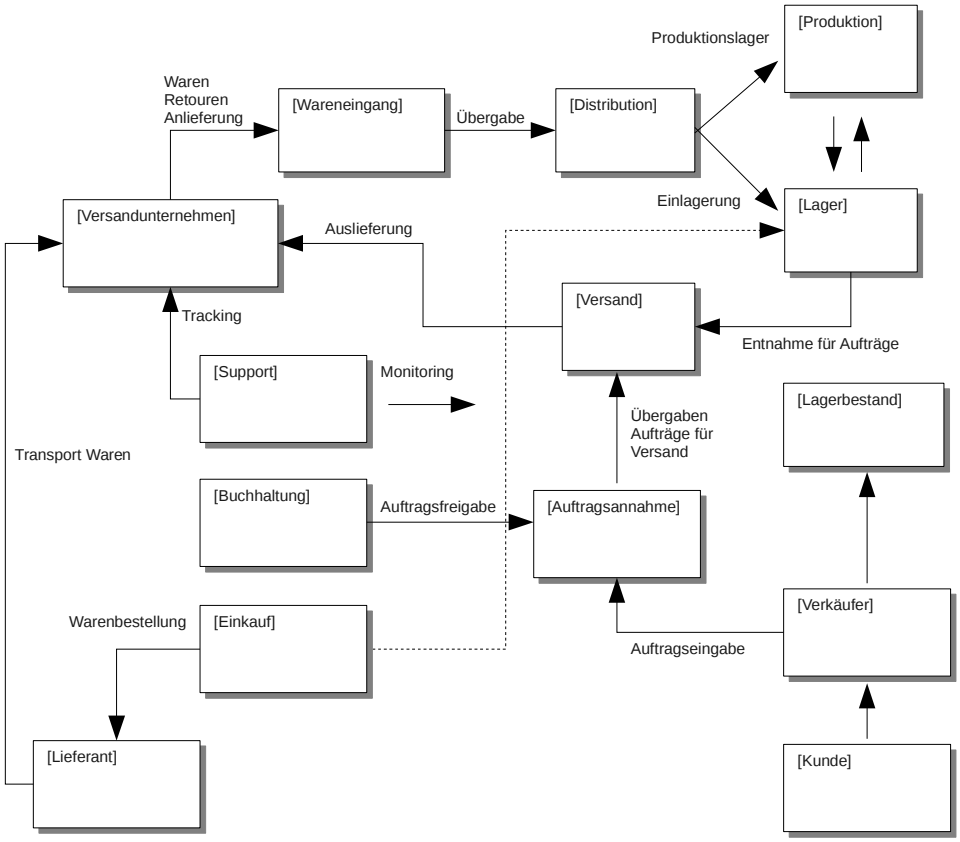


Abbildung 2: Organigramm embedded projects GmbH Geschäftsabläufe

Bereits bei dieser kleinen Firma werden abhängig vom Geschäftsprozess verschiedene Stationen durchlaufen (Wareneingang, Distribution, Lager, Versand, etc.) und somit wird verschiedene Peripherie benötigt. Soll beispielsweise eine Ware für das Lager bei einem Lieferanten bestellt werden, so muss die Bestellung vom Einkauf initiiert werden. Das Versandunternehmen liefert die Ware am Wareneingang an. Nach einer ersten Erfassung per Waage und Kamera wird diese an die Distribution weitergeleitet. Durch die Erfassung des Pakets per Kamera und Waage kann dem Einkauf, der auf die Ware wartet, eine

automatische Meldung gesendet werden. Der Einkäufer weiß so in Echtzeit über den Status der Sendung Bescheid. Für die Lagerware kann in der Distribution für jeden einzelnen Artikel ein Barcodeetikett vom System selbständig ausgedruckt werden. Ein Mitarbeiter muss nur noch das Etikett auf den Artikeln anbringen. Anschließend wird im Lager mittels Barcode-Scanner das Regal und die Ware erfasst, um eine mengenmäßige Zuordnung zum Lagerplatz zu erhalten. Werden Artikel später versendet, so erhält der jeweilige Mitarbeiter über das Warenwirtschaftssystem auf einer separaten Oberfläche die Information, auf welchem Lagerplatz sich welcher Artikel befindet.

Weitere Kombinationen zwischen Geschäftsprozessen und Lager- und Distributionssystemen könnten sein:

- Wareneingang mit Kamera, Waage und Etikettendrucker (siehe oben)
- Prüfstation mit Barcode und automatischem Prüfstand (z.T. auch elektrisch, chemisch oder physikalisch)
- Versandstation mit Barcode-Scanner, Waage, Kamera und Etikettendrucker
- Meldeschalter für LKW-Rampen, Versandunternehmen
- etc.

Auf den ersten Blick sehen die Anforderungen sehr abstrakt und allgemein verwendbar aus. Der zentrale Wareneingang bildet die Schnittstelle zu Lieferanten- und Retourensendungen von Kunden ab. Die Lieferungen müssen erfasst und grob zugeordnet werden. Sind die Lieferungen erfasst, werden diese entweder sofort oder später geöffnet und entsprechend weiterverarbeitet. Für ausgehende Lieferungen existieren Lager bzw. fertige Produktions-Chargen aus „Just-In-Time“ Lieferketten, welche über den Versand zum Kunden bzw. Lieferanten verschickt werden.

Die wesentlichen Kernaufgaben sind in allen Systemen identisch. Doch bei den Implementierungen spielen plötzlich viele Faktoren eine ausschlaggebende Rolle, so dass es im Detail immer individuelle Lösungen sind.

Abhängig von Mitarbeiteranzahl bzw. -qualifikation können bestimmte Aufgaben nur von bestimmten Mitarbeitern erledigt werden. Als Beispiel kann der Wareneingang einer Elektronikfirma betrachtet werden. Regelmäßig kommen Lieferungen mit Bauteilen an. Es muss geprüft werden, ob es sich um die richtige Version der Bauteile (z.T. Abhängig von

sogenannten Day-Codes[1]). Kundenretouren müssen gesichtet werden, wenn es sich um ein defektes Gerät oder um eine Rücksendung, die wieder in das Lager für den Verkauf gelegt werden soll, handelt. Als Mitarbeiter könnte ein Versandfertigmacher, Techniker oder gar ein Ingenieur eingesetzt werden. In der Praxis sieht man tatsächlich an diesen Posten unterschiedlichste Qualifikationen. Abhängig von der Strategie, die gewählt worden ist, muss der Prozess entsprechend definiert und eingerichtet werden.

Für eine einfache Wareneingangsprüfung würde eine Waage und Kamera ausreichen. Sollen bei Retourensendungen Reparaturen in der Firma durchgeführt werden, könnte im Wareneingang nach einer Prüfung bereits das passende Ersatzteil, welches aus einem Lager genommen wird, beigelegt werden, um einen schnelleren und kostengünstigeren Durchlauf zu erreichen.

Das letzte Beispiel soll zeigen, wie unterschiedlich die gleichen Prozesse z.T. sogar intern in Firmen implementiert sein können. Abhängig von diesem Prozess wird eine entsprechende Maske der Anwendung und Konfiguration des Arbeitsplatzes benötigt.

1.2. Datenkommunikation für Geschäftsprozesse

Die Logistik bildet die Grundlage der meisten Geschäftsprozesse. Steht die Logistikkette stabil, so steht der Geschäftsprozess. Die Stabilität des Geschäftsprozesses wirkt sich unmittelbar auf die Ergebnisse einer Firma aus. Stillstand kostet durch die z.T. sehr hohen Taktraten bei Produktionen und Abläufen sehr viel Geld.

Aus diesem Grund sind die Anforderungen an die Komponenten des Netzwerks für die Ansteuerung der Peripherie auch bereits in kleinen Firmen sehr hoch. Unterschieden wird bei den folgenden Anforderungen zwischen den reinen Qualitätsanforderungen „Qualitiy of Service QoS“ und den Dienstgüteanforderungen „Service Level Agreement SLA“.

Qualitätanforderungen „Qualitiy of Service QoS“:

- Schnelle Reaktions- und Zugriffszeiten
- Konstante Performance bei steigender Prozessanzahl

[1] Der Day-Code gibt an, an welchem Tag das Bauteil produziert worden ist. Traceability bzw. Rückverfolgung sind die entsprechenden Stichwörter für diesen Parameter.

- Automatische Fehlerbehandlung bzw. -erkennung

Die schnellen Reaktions- und Zugriffszeiten ermöglichen ein flüssiges Arbeiten. Eine zu große Streuung der Varianz auf den Zugriffszeiten kann Auslöser für Engpässe verschiedenster Art sein: Mitarbeiter warten zu lange auf bestimmte Antworten vom System, die Anzahl der abgearbeiteten Aufträge vermindert sich. Durch eine schnelle bzw. kurze Reaktions- und Zugriffszeit kann diesem entgegen gewirkt werden.

Bei steigenden Prozessen im System muss ebenfalls die Performance des Netzwerkes konstant bleiben. Durch eine entsprechend ausreichende Bemessung der Prozessoren der Komponenten kann die Prozessanzahl bei gleich bleibendem Durchsatz erhöht werden.

Das Netzwerk muss eine automatische Fehlererkennung und gleichzeitig eine Fehlerbehebung für die Komponenten Implementiert haben, um schnell Fehler bzw. entsprechende Lösungen selbst zu finden oder zu melden.

Dienstgüteanforderungen „Service Level Agreement SLA“

Die Dienstgüteanforderungen beschreiben die für den Netzbetreiber mindestens zu erfüllenden Parameter:

- Hochverfügbarkeit
- 24/7h Betriebsbereitschaft
- Einfach zum Austausch bei Defekt
- Einfache Installation
- Gute Verfügbarkeit der Hardware

Bei Störung einer wichtigen Komponente muss ein Ersatzsystem automatisch den Dienst übernehmen. Die Hochverfügbarkeit soll so eine Betriebsbereitschaft von nahezu 100% erreichen. Firmennetzwerke laufen für gewöhnlich 24 Stunden am Tag und 7 Tage die Woche (selten werden Router, Server, etc. in Zwischenzeiten heruntergefahren). Die Komponenten des Netzwerkes für die Schnittstelle müssen ebenfalls für diese Laufzeiten ausgerichtet sein.

Nach dem Ausfall einer Komponente muss per „Plug-and-Play“ ein Austausch möglich sein. Die Erste-Hilfe-Maßnahme soll bestenfalls von allen Mitarbeitern auch ohne tiefe technische Kenntnisse möglich sein.

Alle eingesetzten Hardwarekomponenten müssen auf dem Markt sehr gut erhältlich sein.

1.3. Implementierungen im Einsatz

Wie wird Peripherie aktuell in Systeme integriert - ist die Frage dieses Abschnitts. Im Rahmen der Masterarbeit wurde hierfür eine Marktanalyse bei lokal angesiedelten Firmen durchgeführt. Zum den wichtigsten Firmen gehörten unter anderem ein Fertigungsfirma für elektronische Baugruppen (ca. 500 Mitarbeiter) und eine Service-Unternehmen für Elektrogeräte (ca. 50 Mitarbeiter) welche bereits seit Jahren Peripherie in Abläufe integriert haben. Weitere Informationen stammen aus Fachliteratur [2],[3],[4] und Produktbroschüren [5].

Interne Internetbrowser

Auf mobilen Geräten werden interne Browser als Bedieneinheiten verwendet. Die Webanwendung muss nur mit einer weiteren Oberfläche (angepasst an Displaygröße) erweitert werden. Daten werden über integrierte Tastaturen oder mittels Javascript verarbeitet und an Server gesendet.

WLAN Anbindungen

Die Anbindung mobiler Geräte geschieht meist über ein lokales WLAN-Netzwerk, wenn es „schnurlos“ sein soll. In den Geräten integriert befindet ist ein TCP/IP-Stack, wodurch das Gerät direkt oder mittels Gateways auf Server zugreifen kann. In den meisten Fällen ist ein verschlüsselter Zugriff auf das Netzwerk bzw. der Aufbau einer SSH oder SSL-Verbindung möglich.

HID[2] Implementierungen (z.B. Barcodescanner per Cursor)

Barcodescanner oder moderne „Pick-by-Voice[3]“ Mikofone geben die Daten an der aktuell gesetzten Cursorposition aus. Mittels Javascript kann der Datenstrom entsprechend manipuliert oder verarbeitet werden.

Ausführbares Programm

Sind für das Gerät Treiber oder ausführbare Dateien vorhanden, können diese mittels externem Befehlsaufruf in die eigene Software integriert werden.

Webservices

Per Webservice bieten diverse Anwendungen eine Kommunikation für Server, Clients und Peripherie an.

SPS Schnittstellen

Die grafische Programmierung mittels SPS[4] ist für aufwändigere Abläufe, bei denen zum Teil mechanische Prozesse integriert sind, ein oft eingesetztes Mittel. SPS-Systeme gibt es fertig konzeptioniert in Modulbauformen, welche einen flexiblen Einsatz ermöglichen.

Java Applets auf mobilen Geräten

Handhelds, Funk-Barcodescanner, Inventurerfassungsterminals, etc. sind ähnlich wie moderne Mobilfunkgeräte direkt mit einer integrierten JVM[5] in der Lage, Java Anwendungen selbst auszuführen. Durch die Integration der Anwendung in einen

---

2 Human Input Device bezeichnet die Gruppe aller Tastatur- und Mausähnlichen Eingabegeräte

3 Befehl von der Anwendung werden per Kopfhörer an den Mitarbeiter übergeben, bzw. kann der Mitarbeiter ebenfalls Antworten über ein Mikrophon an die Anwendung mittels Spracherkennung senden. Die Hände bleiben so für den eigentlichen Prozess frei.

4 Speicherprogrammierbare Steuerung ist eine verbindungsorientierte Programmierung von Steuer- und Regelungen

5 Java Virtual Machine

Serverdienst sind Geräte mittels RPC[6] direkt aus der eigentlichen Anwendung transparent aufrufbar.

Feldbussysteme mit der speicherprogrammierbaren Steuerung

Verschiedenste Feldbussysteme aus der Industrieautomatisierung kommen bei Installationen zum Einsatz. Entschieden wird hier primär nach den Faktoren, welche Feldbusse bereits vorhanden sind, was wird an Schaltungen (Sensoren, Aktoren) auf dem Mark angeboten und wie sehen die Faktoren Leistungsfähigkeit, Stabilität und Kosten im Vergleich aus? Eine Liste typischer Feldbusse kann [6] entnommen werden.

1.4. Warehouse-Managment-Systeme WMS

Der Bereich des „Warehouse-Management WMS“ beschäftigt sich, wie in Abbildung 3 (Grafik vom Fraunhofer Institut für Materialfluss und Logistik) gezeigt, mit den Kernfunktionen Auftragsbearbeitung, Auftragsfreigabe, Bestandsführung, Inventur, Informationssysteme, Stammdaten, Wareneingang, Einlagerung, Lagersteuerung, Kommissionierung, Auslagerung und Warenausgang. Abhängig von den Kernfunktionen gliedern sich weitere Randthemen wie z.B. Retouren, Ressourcenplanung u.v.m. an die Thematiken.

Basierend auf dem WMS können sich übergeordnetere Schichten wie ERP-[7], WWS[8], SCM-[9], PPS[10] und CRM-Systeme[11] befinden. Betrachtet man die Schichten unterhalb des WMS[12] so befindet man sich sehr schnell in einer konkreten Hardwareebene. TMS[13], MFR[14] und SPS sind die zu Grunde liegenden Mechanismen um Prozesse, welche mit Hardwareanbindung implementiert sind, anzusteuern.

---

6 Remote Procedure Call; Ein entfernter Funktionsaufruf ermöglicht den Aufruf einer Funktion eines Programms auf einem entfernten Prozessor. Der Aufruf wird aber so in die Entwicklungsumgebung bzw. das Programm eingebettet, dass es für den Entwickler wie ein interner Funktionsaufruf aussieht.

7 Enterprise Resource Planning: Planung der Unternehmensresourcen

8 Warenwirtschafts-System

9 Supply-Chain-Management: Lieferkettenmanagement

10 Produktinsplanungs- und Steuerungssystem

11 Customer-Realtionship-Management: Kundenbeziehungsmanagement

12 Warehouse-Management-System: Lagerverwaltungssystem

13 Transport-Management-Software: Disposition-Tourenplanungs-Software

14 Materialflussrechner

Der Bereich des Warehouse-Management ist ein Zusammenspiel zwischen klassischen Themen wie Operations-Research, Künstliche Intelligenz und vielen weiteren Informatikgebieten wie Datenbank-Management-Systemen, Serviceorientierte Architekturen etc. [2]

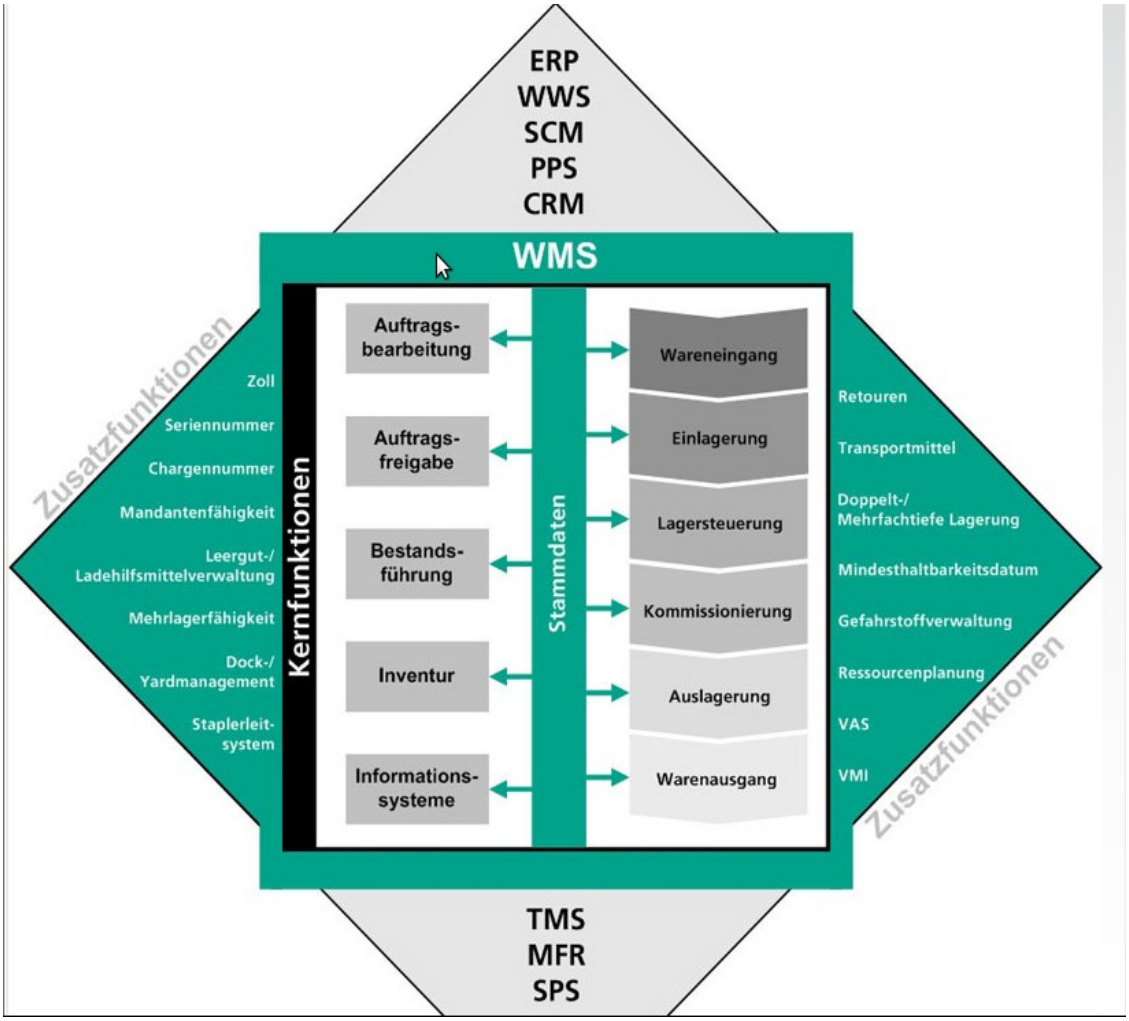


Abbildung 3: Themengebiete WMS [7]

1.5. Webtechnologien / Internetstandards

Abhängig von Programmiersprachen und Betriebssystemen gibt es für Webtechnologie eine große Auswahl. Tabelle 1 beinhaltet mögliche Kandidaten für den Zugriff mittels Software über Netzwerke auf Schnittstellen.

Technologie	Betriebssystem abhängig	Typische Anwendungen	Standards
RPC	hetrogen	Speziallösungen, Interne Anwendungen, Ressourcensparende Kommunikation	RFC 707, RFC 1057, RFC 1831 [8],[13]
SOAP	hetrogen	Typische Webdienste (Amazon, Ebay, etc.)	W3C [9]
DCOM	Fokus Windows	Lokale Steuerungen	Microsoft [10]
CORBA	hetrogen	Große Systeme	OMG Group [11]
REST	hetrogen	Lose gekoppelte Anwendungen	Fielding Roy [12]

Tabelle 1: Webtechnologien

Für das Netzwerk der Masterarbeit wurde die Technologie REST per HTTP ausgewählt. Eine REST Implementierung ist mit jeder Programmiersprache möglich, welche die Programmierung von Netzwerk-Sockets anbietet oder als CGI [15]ablaufen kann.

Genaueres kann dem Kapitel „3. REST Implementierung“ entnommen werden.

1.6. SIRENA und DPWS

Seit Juni 2009 existiert eine Spezifikation DPWS „Device Profile for Web Services“, welche die Ansteuerung von Geräten über einen Webservice (SOAP) beschreibt. Als erste Implementierung bietet SIRENA „Service Infrastructure for Real-time Embedded Networked Devices“, eine Schnittstelle für DPWS an. Das Projekt steht unter einer Open-Source Lizenz zum freien Download zur Verfügung. Die Spezifikation von DPWS kann aus dem Internet geladen werden [21]. Die Software von SIRENA ist in Java geschrieben. Im

[15] Common Gateway Interface, Schnittstelle um Programme über Webserver ansteuern zu können.

Vergleich zu der in der Masterabeit entwickelten Lösung basiert das Zusammenspiel der Komponenten unter DPWS vollständig auf den Kern-Technologien für Webservices: WSDL 1.1, XML-Schema, SOAP 1.2, WS-Addressing, WS-MetadataExchange, WS-Transfer, WS-Policy, WS-Sercurity, WS-Discovery und WS-Eventing. In der Masterarbeit wurden typische Mechanismen (DHCP, DNS, ARP, HTTP, etc.) aus der Netzwerkwelt verwendet, welche einen erheblich geringeren Aufwand für die Lösung auf Mikrocontrollern hinsichtlich Prozessorleistung und Implementierungszeit bringen [20] [21] [22].

1.7. Marktanalyse

Im Folgenden sind Firmen für Warenwirtschaftssysteme zusammengetragen. Keines der gefunden Produkte bietet eine freie und flexible Schnittstelle für Hardwareanbindung an.

LingoistiX GmbH

Die LingoistiX GmbH [14] ist ein Spin-Off des Fraunhofer Instituts für Materialfluss und Logistik. Die Firma vertreibt und implementiert ein Open-Source Programm Namens myWMS. Haupttechnologie für Embedded Übertragungen ist bei LingoistiX RFID[16]. Als Leser wird ein Gerät mit dem Namen ePOD eingesetzt. ePOD ist ein tragbares Gerät mit Tastatur, Display und integriertem Browser, welcher über ein Display sichtbar ist.

Mad geniuses GmbH

„pixi“ ist eine umfassende Software für den Online-Versandhandel, die alle Prozesse in einer geschlossenen Prozesskette vereint - effizient, zuverlässig und aus einer Hand.

Eine Integration von Barcode-Scannern und Etikettendruckern ist in der Software (Windows Anwendung) über das Betriebssystem realisiert. Das Gerät muss an einem Rechner angesteckt sein, von dem aus der Prozess gesteuert wird. Entferne Lösungen werden über Netzlaufwerke und Druckerfreigaben gesteuert.

[16] RFID Radio Frequency Identification. Technologie für Funkübertragung von Kennungen

ELV-Systemtechnik

Speedypick von ELV Systemtechnik ist ein kleines Display mit zwei Tastern, welches an Transportkisten oder Regale gesteckt werden kann. Die Ansteuerung erfolgt über einen RS485 Bus. Für externe Software werden Treiber angeboten, die jedoch nur als Dienstleistung von der Firma ELV-Systemtechnik integriert werden können.

Openbravo

OpenBravo ist ein ERP Open-Source Projekt betreut durch die Firma *Openbravo, S.L.* Die in Java entwickelte Anwendung basiert auf einer Oracle oder Postgre Datenbank. Eine Hardwareanbindung gibt es zum jetzigen Zeitpunkt noch nicht.

Weitere Firmen und Produkte

Firma	Produkt
inconso AG	inconsoWMS
GIGATON GmbH EDV- und Netzwerkberatung	LogoS C/S
prismat GmbH	SAP EWM
Coglas GmbH	Coglas
InnoLOG GmbH	MoTIS® LSV
Aberle Automation GmbH & Co. KG	PMS-L
Dr. Thomas + Partner GmbH & Co. KG	TWS
TGW Systems Integration GmbH	CI_WMS
ISA - Innovative System Solution for Automation	ISASTORE
Salomon Automation GmbH	WAMAS
SALT Solutions GmbH	SAP EWM / LES TRM
CIM GmbH Logistik-Systeme	PROLAG® World
S&P Computersysteme GmbH	SuPCIS-L
XELOG AG	XELOG LagerSuite

Tabelle 2: Übersicht Produkte [7]

Im nächsten Kapitel wird die eigene Lösung erst abstrakt dargestellt und in späteren Kapiteln konkret bis hin zur Implementierung bearbeitet.

2. Architektur der eigenen Schnittstelle

Das Konzept der gesamten Schnittstelle bzw. des Netzwerks für die eigene webbasierte Ansteuerung von externer Peripherie bedarf mehrerer Soft- und Hardwarekomponenten. Im Folgenden wird auf die benötigten Module und deren Aufgaben im Verbund eingegangen. Die technischen Details der einzelnen Implementierungen werden in den nachfolgenden Kapiteln der Masterarbeit beschrieben.

Die Masterarbeit beschreibt und implementiert ein vollständiges Netzwerk (Protokoll, Geräteverwaltung, Adressierung, Management, Serverdienst, Client-Bibliothek, etc.) - welches zum Teil auf bestehenden Technologien – und zum Teil als eigene Implementierungen realisiert worden ist.

2.1. Anwendungen für webbasierte Steuerungen

Immer dann, wenn Peripherie bzw. Geräte aus einer Webanwendung heraus genutzt werden sollen, ist das Netzwerk ein passendes Framework. Abbildung 4 zeigt eine typische Konfiguration einer solchen Anwendung.

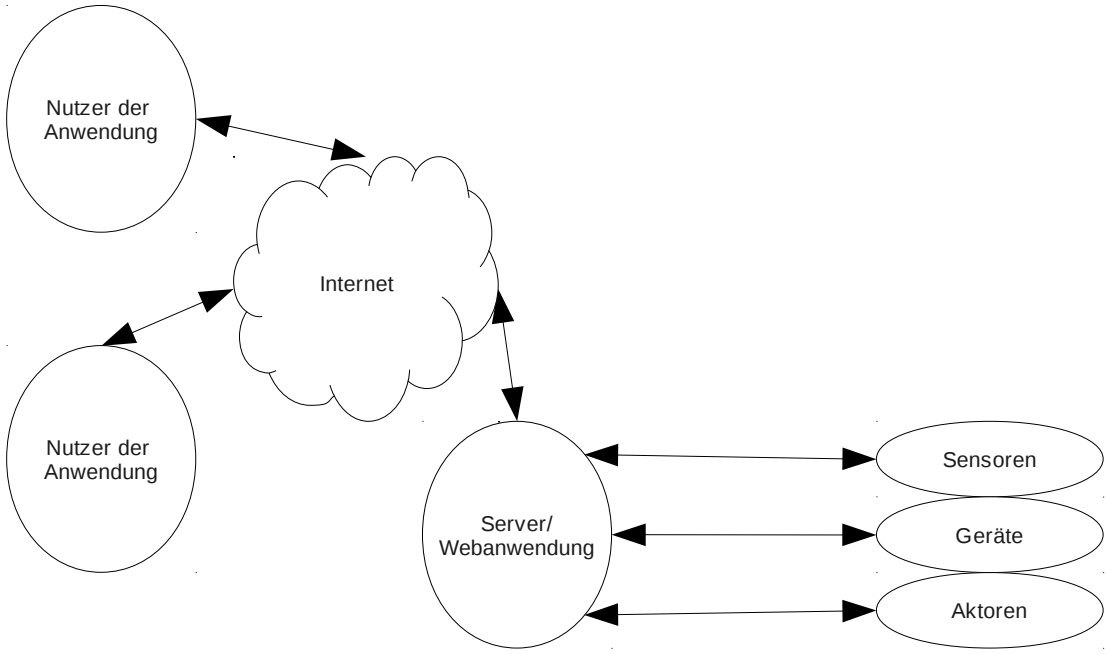


Abbildung 4: Anwendung für webbasierte Steuerung

Sensoren, Geräte oder Aktoren sollen in den Programmfluss der Webanwendung mit aufgenommen werden, so dass diese für die Benutzer zur Verfügung stehen und visuelle Ausgaben oder Datenbankeinträge erzeugt werden.

Typischer Anwendungen:

- Warenwirtschaftssystem mit Logistikprozessen
- Automatische Lager
- Online-Heizungssteuerung
- Online-Wetterstation
- Online-Alarmanlage
- etc.

Ein Problem von Netzwerken für Prozesse, bei denen viel Peripherie und viele Computer eingebunden sind, ist der Stromverbrauch. Lässt man Serveranwendungen zum größten Teil 24 Stunden 7 Tage pro Woche laufen, so besteht bei Arbeitsplatzrechnern eher das Bedürfnis, die Computer nur während der Arbeitszeiten eingeschaltet zu haben. Soll auch jedes periphere Gerät regelmäßig aus- und eingeschaltet werden, wird ein zentrales Power-Management notwendig. Alle Prozesse müssen auf Standby sein und einfach bei Bedarf oder Arbeitsbeginn aktiviert und anschließend wieder deaktiviert werden können (siehe Abbildung 5).

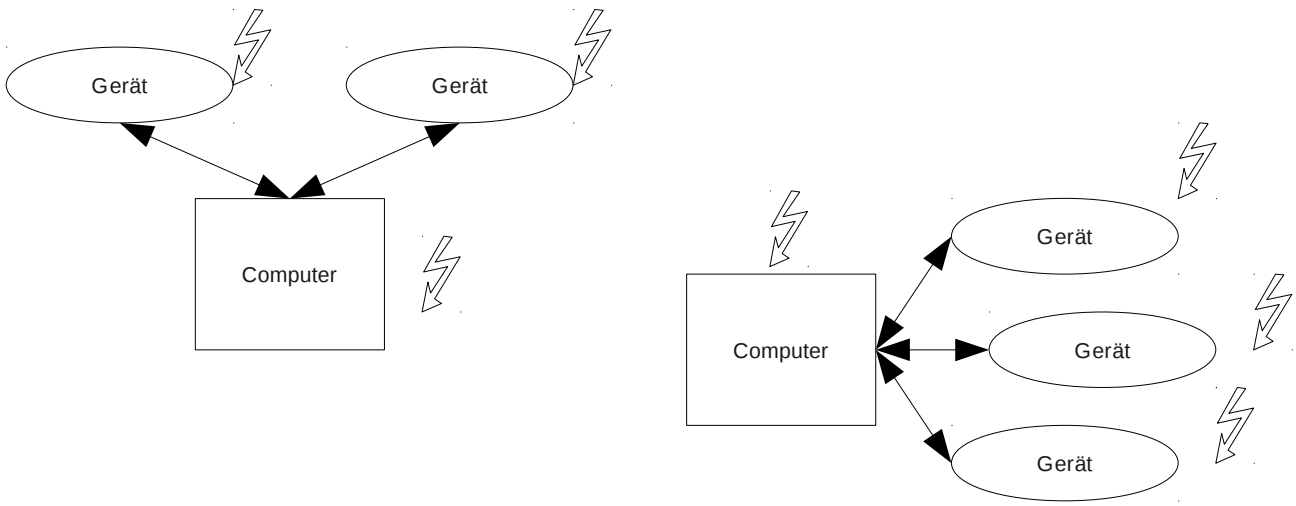


Abbildung 5: Stromverbrauch Netzwerk

Zusammengefasst bietet das Netzwerk folgende Leistungen vollständig webbasiert an:

- Anbindung von externer Peripherie an webbasierte Anwendungen
- Power-Management für Netzwerke (für Computer und externe Peripherie)

2.2. Komponenten im Überblick

Abbildung 6 zeigt eine typische Konfiguration des Netzwerks mit allen beteiligten Komponenten. Zentrale Komponente ist die sogenannte Management-Konsole, welche zu allen Adapter-Boxen (Schnittstelle zu Geräten) eine Verbindung hat. Ein Arbeitsplatz ist typischerweise oft mit einer Vielzahl verschiedenster Geräte, Sensoren und Aktoren ausgestattet.

Für die Anwendung ist die Management-Konsole ebenfalls der primäre Kommunikationspartner. Über die Konsole können Geräte angefordert und allokiert werden. Nach der initialen Kommunikation mit dem Zielgerät kann die Anwendung eine direkte Punkt-zu-Punkt Verbindung aufbauen.

Eine Gruppierung von Geräten oder Rechnern (optional auch mehrfach in Kombination) wird als Station bezeichnet. Für das Netzwerk ist die Kenntnis über eine Gruppe nur für das Power-Management notwendig, um beim ein- oder ausschalten einer Station alle angeschlossene Geräte ein- und auszuschalten bzw. zu allokieren.

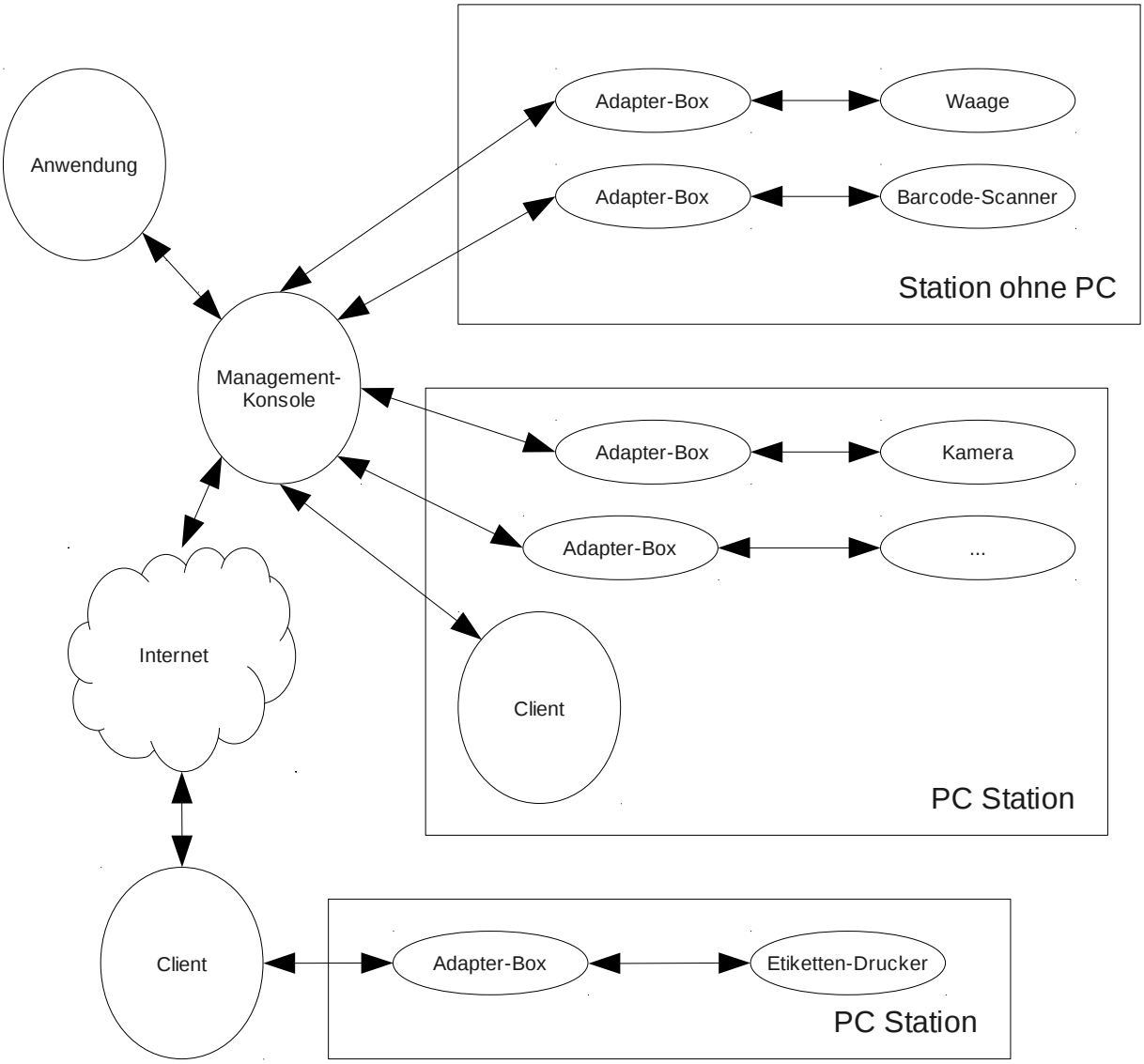


Abbildung 6: Blockdiagram Netzwerk

Externe Clients können über das Gateway-Interface der Management-Konsole in das interne Netzwerk und so die Geräteverwaltung integriert werden.

2.3. Adapter-Box

Um Peripherie wie z.B. Barcodescanner, Waagen und Etikettendrucker an webbasierten Anwendungen Betriebssystemunabhängig betreiben zu können, dient eine kostengünstige im Rahmen der Masterarbeit entwickelte Adapter-Box (siehe Abbildung 7). An der Adapter-Box können per USB, RS232 und vielen weiteren typische Mikrocontroller-Schnittstellen externe Geräte angeschlossen werden. Der Treiber wird vom Betriebssystem in die Adapter-Box ausgelagert. Über einen LAN-Anschluss wiederum bietet die Adapter-Box die Funktionen des Geräts in einem angeschlossenem Netzwerk an.

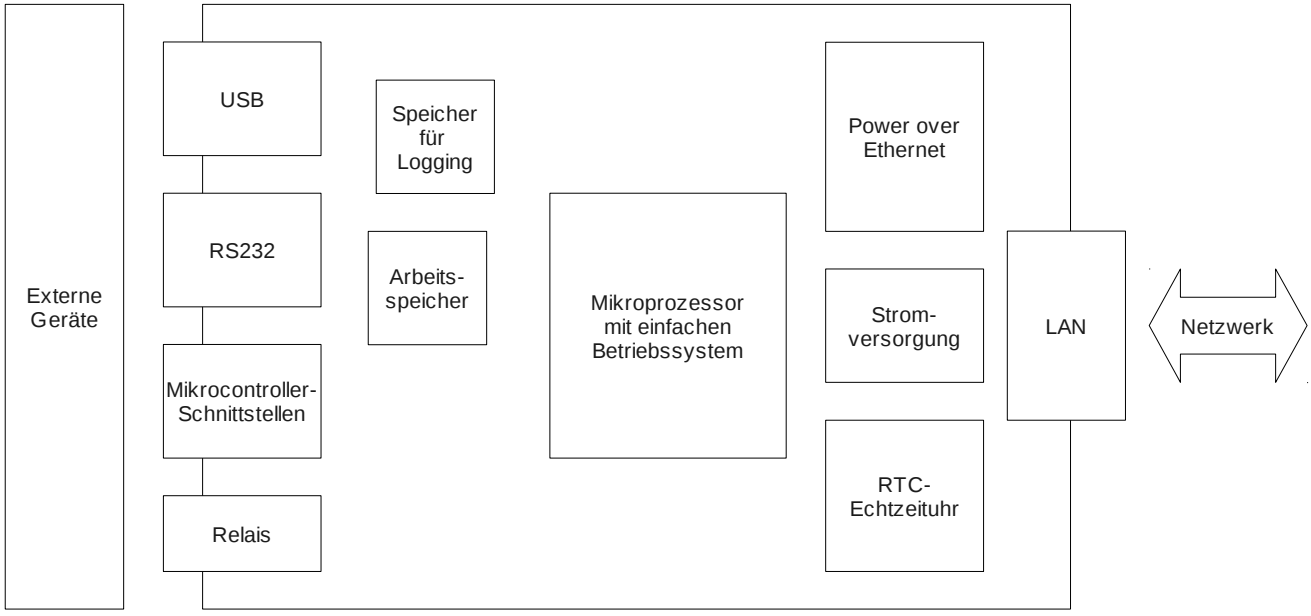


Abbildung 7: Adapter-Box für Anschluss externer Geräte

Jede Adapter-Box meldet sich nach dem Einschalten automatisch an einer zentralen Management-Konsole an, um vom Netzwerk aus angesteuert werden zu können.

2.4. Management-Konsole

Die Hauptaufgaben der Management-Konsole sind die Verwaltung der Verbindungen zu Adapter-Boxen und der Betrieb des zentralen Zugriffsknotens für die Anwendung. Des weiteren fallen Aufgaben wie z.B. die Verwaltung für das Netzwerk (Teilnehmerlisten, Geräteverwaltung, Logdateien von Adapter-Boxen abholen, Sicherung der Hoch-Verfügbarkeit, Implementierung eines Gateway-Interfaces, etc.) an.

Das zentrale Power-Management ist ebenfalls Aufgabe der Management-Konsole. Zeit- und Ereignisgesteuert müssen Teilnehmer (Rechner und Geräte) ein- und ausgeschaltet werden. Hierfür müssen verschiedene Techniken und Technologien wie z.B. Power-over-Ethernet, Wake on LAN, Ansteuerung externer USB-Steckdosen, etc. genutzt werden.

Die Management-Konsole ist somit:

- Schnittstelle zwischen der Anwendung und der externen Peripherie
- Schnittstelle und Zentrale für das Power-Management

Eine weitere Anforderung an die Management-Konsole ist, dass eine Schnittstelle für typische grafische Verwaltungsoberflächen (siehe Abbildung 8) angeboten werden muss. Das heißt, dass mit einfachen Mitteln die Funktionen der Konsole grafisch ansteuerbar und visualisierbar, unabhängig von der eingesetzten Programmiersprache oder Technologie (Webanwendung, Lokale Anwendung, etc.) in die Verwaltungssoftware (z.B. Warenwirtschaftssystem) verwendet werden können sollen.

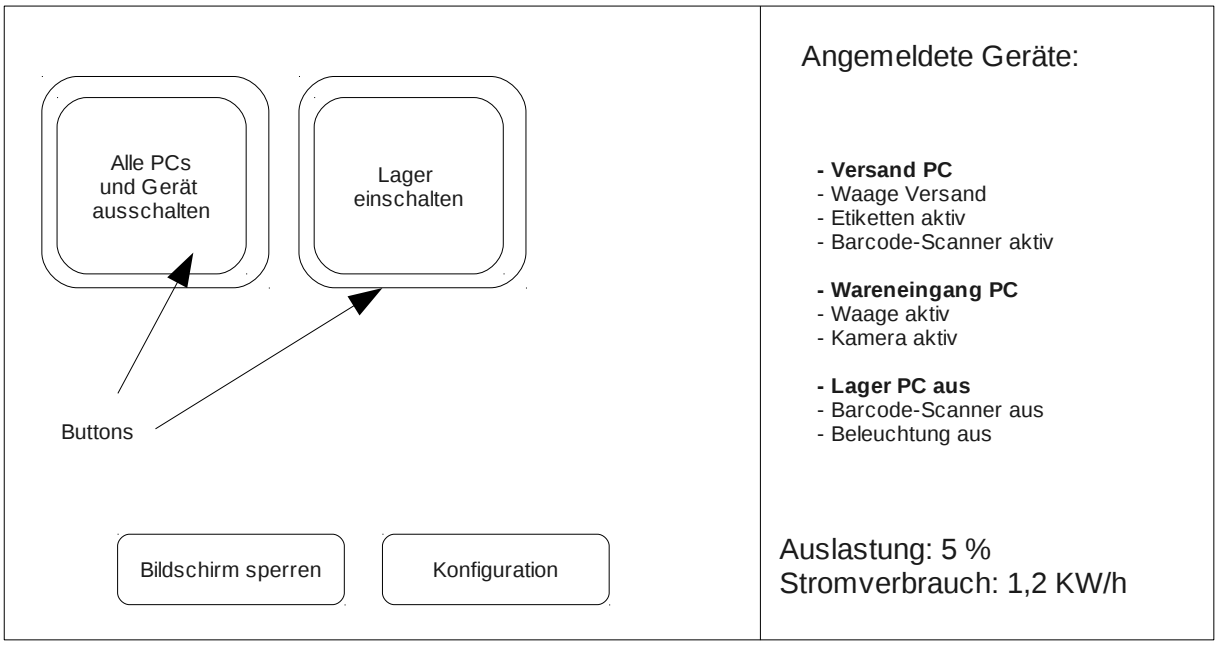


Abbildung 8: Oberfläche Managment-Konsole

Die Management-Konsole kann als Programm auf einem bestehenden Server der Firma installiert, in eine eigene Hardware (ähnlich wie klassische Router oder Firewalls) integriert, oder in der besonderen Ausführung mit einem Display und Touchscreen als typischen Panel-PC für die Wandmontage eingebaut sein (siehe Abbildung 9).

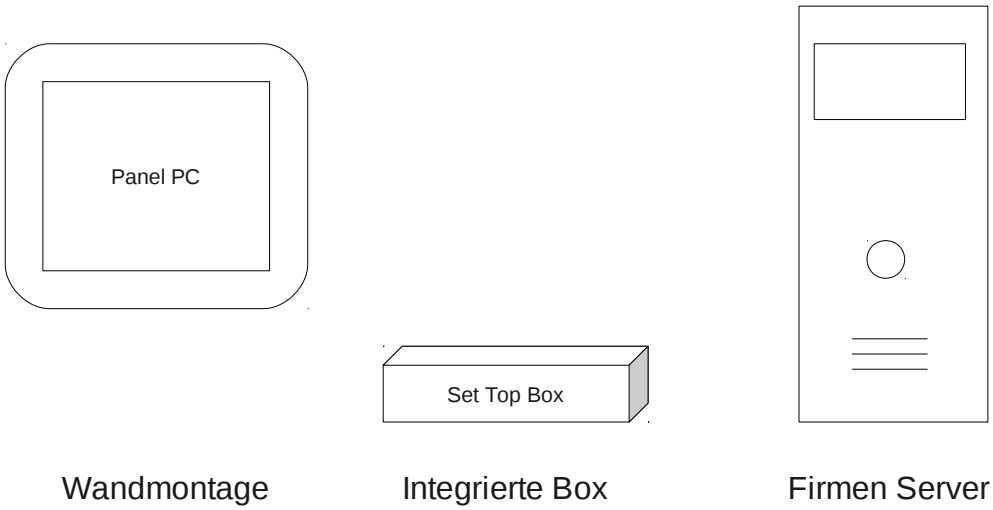


Abbildung 9: Integration Management-Konsole

2.5. Power-Management

Als weiteres Ziel wurde ein vollständig integriertes Power-Management implementiert. Vollständig heißt, dass wirklich jeder teilnehmende Computer integriert wurde. Ziel ist es, jedes im Netzwerk des Warenwirtschaftssystems hängende Gerät von der Waage bis zu den beteiligten Computern über das Netzwerk als Webservice-Aufruf gezielt ein- und ausschalten zu können.

Techniken wie „Wake on LAN“, „Power over Ethernet PoE“, eine über die Adapter-Box ansteuerbare 230V Steckdose (Kapitel 6.7. USB-Steckdose für Power-Management), die Relais für externe Ansteuerungen (Kapitel 6.7. USB-Steckdose für Power-Management), die Echtzeit Uhr für zeitgesteuert Triggerung und der Stations-Daemon für die Einbindung

der Computer auf die gleiche Art und Weise wie die Adapter-Boxen, kommen hier zum Einsatz.

Weitere Informationen zu den Themen gibt es in Kapitel 4. Struktur und Aufbau der Adapter-Box und 5. Softwarearchitektur der Management-Konsole .

2.6. Ausfallsicherheit bei Stromausfall mit Logdatei

Alle Transaktionen werden in der Adapter-Box auf einer microSD-Karte direkt nach dem Eintreffen gespeichert. Nach erfolgreicher Abarbeitung wird der Eintrag entsprechend markiert. Mit diesem Mechanismus kann die Adapter-Box nach einem plötzlichen Stromausfall feststellen, ob ein Kommando wiederholt bzw. der Ausfall gemeldet werden soll. Parallel wartet der Sender auf die Antwort bzw. erhält keine Antwort nach einem Timeout und kann dies entsprechend in der Anwendung visualisieren.

2.7. Begriffe

Definiert werden nun die verwendete Bezeichnungen für die einzelnen Komponenten, um in der Masterarbeit und Software eine einheitliche Sprache als Basis zu haben:

Bezeichnung	Komponenten	Bemerkung
Adapter-Box	Hardwareschaltung für Anbindung der Geräte an das Netzwerk	Kapitel 4.
Management-Konsole	Software für Verwaltung und Betrieb des Netzwerkes	Kapitel 5.
Anwendung	Die Software welche das Netzwerk verwendet bzw. integriert hat	In der Masterarbeit ist es das Warenwirtschaftssystem waWision 2.15
Station	Zusmmanestellung von Komponenten	
Stations PC / Computer	Eingebundene Computer (Softwaretechnisch identisch eingebunden wie Adapter-Box)	Wird realisiert durch einen Hintergrund Prozess auf den Computer Kapitel Fehler: Referenz nicht gefunden
Label	Bezeichnung eines Teilnehmers (Adapter-Box oder Stations PC)	Kann über API verändert werden

Tabelle 3: Bezeichnungen

3. REST Implementierung

Der Markt der Technologien für die Implementierung von Web-Services ist vielfältig. Zwischen RPC, SOAP, etc. gibt es jedoch einen weit verbreiteten jedoch noch fast unbekannten Architekturstil: Eine REST-orientierte Implementierung auf der Basis des HTTP-Protokolls.

Warum wurde REST gewählt? REST ist ein Architekturstil, für den sich das HTTP-Protokoll als sehr gute Beispielimplementierung bewährt hat, bzw. REST inspiriert von diesem entwickelt wurde [12]. REST mach keine Vorgaben für technische Implementierung.

Der entscheidende Faktor für die Wahl von einem REST orientierten Ansatz über primär HTTP ist, dass bereits für kleinste Mikroprozessoren sehr umfangreiche TCP/IP Stacks existieren, mit welchen ein passender REST-Server ohne viel Aufwand implementiert werden kann. Desweiteren lassen sich REST-Aufrufe sehr gut kapseln und so über weitere typische Mikrocontrollerprotokolle an andere Netzwerke oder Kommunikationspartner senden.

Alternativ zu dem REST-orientierten Ansatz wäre eine einfache XML-RPC Implementierung mit ähnlichen minimalen Ressourcen möglich gewesen. Alle weiteren Netzwerktechnologien benötigen jedoch einen wesentlich höheren Ressourceneinsatz und sind dafür für kleine Embedded-Systeme wie die Adapter-Box nicht geeignet.

Die typischen Schlagwörter für einen REST-orientierten Ansatz sind:

- REST ist einfach
- REST ist weit verbreitet
- REST ist flexibel
- REST ist eine lose Kopplung
- REST bietet Interoperabilität
- REST ermöglicht immer eine echte Wiederverwendung

3.1. Einführung in REST

Folgender Abschnitt gibt einen kurzen und groben Einblick in den Architekturstil REST. Weitere Informationen können der Literatur [12] [16], sowie zahlreichen Internetseiten entnommen werden.

Die beiden wichtigsten Faktoren bei REST sind:

- Der Server hat keinen Zustandsspeicher sondern nur Operationen (Der Zustand wird immer durch den Client dargestellt)
- Die Namen und Anzahl der Operationen sind für jede Anwendung gleich (REST-Standard)
- Ressourcen bilden die Schnittstelle zur Anwendung

Beim Entwicklungsprozess von REST-orientierten Anwendungen müssen Anfänger regelmäßig Checklisten durchlaufen, um nicht unbewusst in einen objektorientieren Ansatz zu geraten. Ein guter Vergleich zwischen der klassischen Architektur und einem REST-basierten Ansatz stammt aus dem Buch REST und HTTP von Stefan Tilkov [16]. Die Abbildung 10 wurde aus dem Buch entnommen und etwas erweitert.

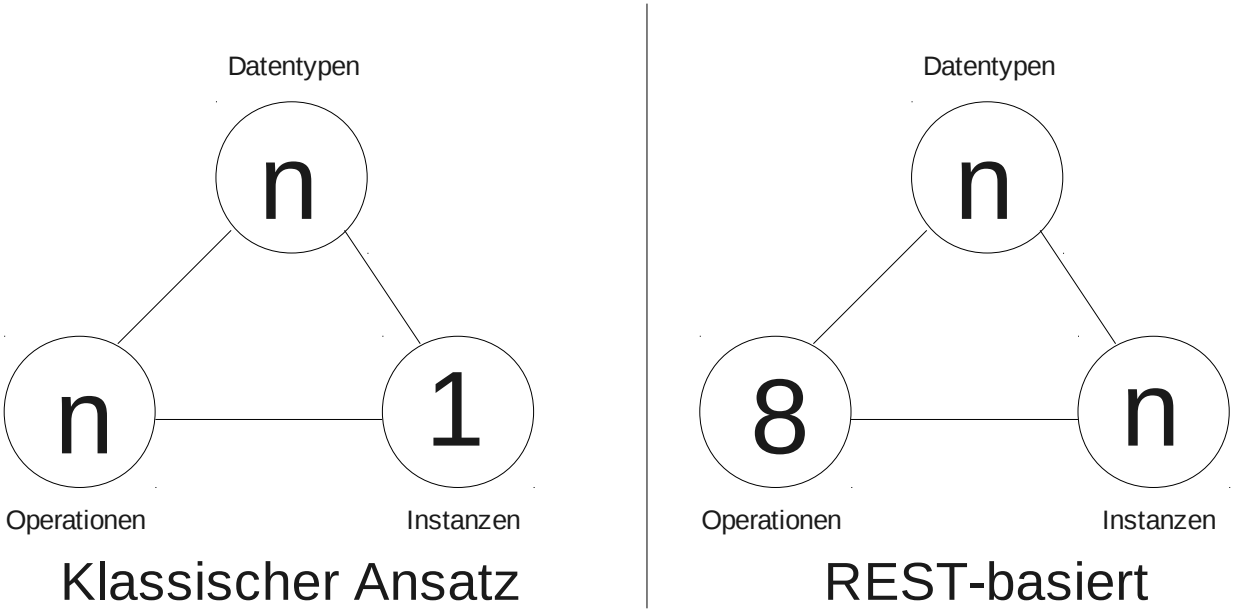


Abbildung 10: Klassischer Ansatz vs. REST-basierter

Im klassischen Entwicklungsansatz werden erst die Objekte - sprich die Datentypen - gesucht und dann die entsprechen Operationen bzw. Methoden. In der Regel gibt es für jeden Webservice eine Instanz.

Im REST-basiert Ansatz sind die Operationen bereits definiert. Sie stammen aus dem HTTP-Protokoll und lauten: GET, POST, PUT und DELETE. Zudem gibt es noch vier weitere, die aber eher selten Einzug in Anwendungen finden: HEAD, OPTION, TRACE und CONNECT.

Die Operation wird direkt mittels HTTP auf einer Ressource (einer URI) ausgeführt. Ressourcen sind meist die Substantive aus der Beschreibung der Anforderung der Software. Jede Ressource ist über eine eindeutige URI erreichbar.

Der Einsatz des HTTP-Protokolls bedeutet, dass alle Anwendung aus diesem Bereich verwendet werden können: Clients wie Curl, Proxys, Loadbalancer, Cluster, Gateways, etc.

Zurück zu den Unterschieden zum klassischen Ansatz für den Anwendungsentwickler: Das Wesentliche ist, dass nur noch an der Anzahl der Parameter „Datentypen“ und „Instanzen“ gedreht werden kann. Dies äußert sich so, dass es für den Zugriff auf einen Kunden im klassischen Ansatz eine zentrale Funktion geben würde, hingegen im REST-basierten Ansatz würde es so viele URIs geben wie viele Kunden es gibt.

Eine einfache Taskverwaltung könnte wie in Tabelle 4 aufgelistet realisiert sein.

Resource	**URI**	**Methode**
Liste aller Tasks	/tasks	GET, POST
Einzelner Task	/tasks/{id}	GET,PUT,POST
Laufende Tasks	/tasks/?state=running	GET
Beenden eines Tasks	/tasks/{id}/cancel	GET

Tabelle 4: Taskverwaltung

In der REST-basierten Lösung sind das die sogenannten Methoden, was in HTTP als Verben bezeichnet wird. Im Folgenden wird kurz auf die einzelnen Verben eingegangen:

- **GET** Diese Operation dient dazu, Informationen von einer URI abzuholen
- **HEAD** Entspricht GET, nur werden ausschließlich die Kopfdaten zurückgegeben

- **PUT** Aktualisiert oder legt eine Ressource an
- **POST** legt eine Ressource an oder übergibt beliebige Kommandos an den Server
- **DELETE** löscht eine Ressource
- **OPTIONS** liefert Metadaten über eine Ressource
- **TRACE und CONNECT** dienen zur Diagnose von HTTP-Verbindungen

3.2. Repräsentationsformat

Das Repräsentationsformat ist die Formatierung der Daten. Von klassischem HTML über XML, CSV usw. bis hin zu speziell angepassten Mikroformaten für Kalender, Adressen existieren viele Formate.

Bei Standardformaten wird das Format als MIME Datentyp (RFC-822, RFC-2045 – RFC 2049) angegeben.

Für die Ansteuerung von Geräten über das Netzwerk wurde ein einfaches eigenes Format definiert (text/device), welches speziell auch mit wenig Rechenleistung und Speicher auf dem Mikrocontroller direkt verarbeitet werden kann.

Abbildung 11 zeigt den Aufbau eines Frames.

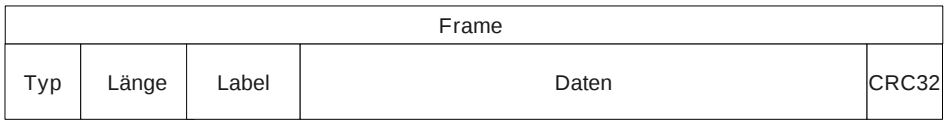


Abbildung 11: Frame für Kommunikation

Ein Frame ist wie folgt aufgebaut: Ein Typ-Feld gibt an, um was für ein Frame es sich handelt. Mit der anschließenden Länge kann der Empfänger ermitteln, wie viele Werte empfangen werden müssen, bis die Nachricht vollständig ist. Das Label gibt dem Empfänger der Nachricht an. Die Pakete werden primär in HTTP-Nachrichten verpackt direkt an den Empfänger gesendet. Mit dem Label ist jedoch auch ein internes Routing zwischen den Adapter-Boxen und der Management-Konsole möglich. Das Datenfeld dient

ausschließlich zur Übertragung der Ergebnisse oder Zustände. Das Datenfeld beinhaltet keine Kommandos oder Parameter. Diese werden vollständig per HTTP abgebildet. Im Datenfeld befinden sich nur die Werte, welche im Body einer HTTP-Nachricht übertragen werden (siehe Abbildung 12). Die CRC32 Prüfsumme ermöglicht eine einfache Überprüfung der Datenübertragung.

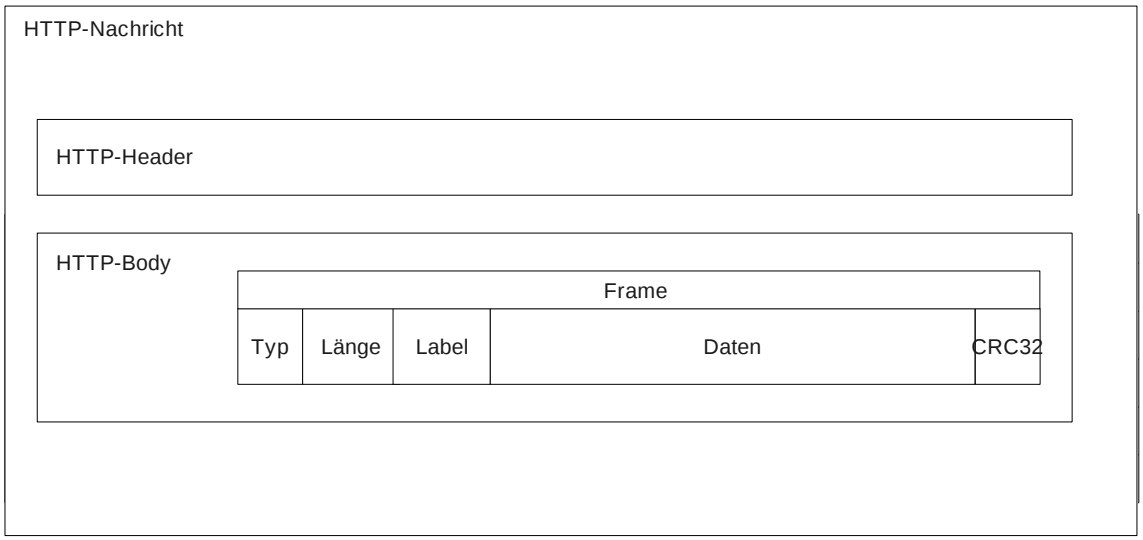


Abbildung 12: Integration Netzwerk-Frame in HTTP-Nachricht

Das Typfeld (Tabelle 6) beschreibt, um was für ein Frame es sich handelt. Abhängig von diesem kann der Empfänger schnell entscheiden, mit welcher Routine die Nachricht bearbeitet werden muss.

Typ	**Wert**	**Beschreibung**
REQUEST	0x01	Standardanfrage an ein Gerät
ANSWER	0x02	Anwort aufgrund einer Anfrage von einem Gerät
EVENT	0x03	Beliebige Asynchrone Events ohne Standard REQUEST ANSWER Prozess

Tabelle 6: Typ Optionen

Die Abarbeitung eines Frames im Empfänger ist in Abbildung 13 dargestellt. Nach dem Empfang der vollständigen Nachricht (Ermittlung der Länge über Längenfeld, welches als

erstes empfangen wird) wird die CRC Prüfung gestartet. Nach der erfolgreichen Berechnung startet die Auswertung mit anschließender Abarbeitung der Anfrage. Das Ergebnis wird entsprechend wieder zusammen gestellt, eine CRC Prüfsumme berechnet und eingefügt. Anschließend wird das Paket dem Sender aus dem Sendepuffer übermittelt.

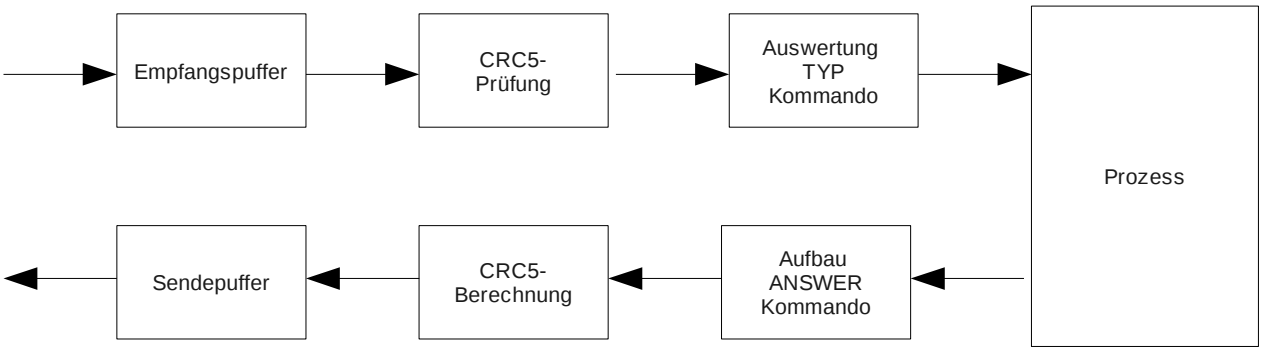


Abbildung 13: Verarbeitung eines Frames

3.3. Physikalische Übertragung

Empfangen wird das Frame immer über das Netzwerk eingebettet in HTTP per TCP/IP. Es können sich jedoch weitere Netzwerke hinter den Adapter-Boxen (siehe Abbildung 14) befinden. Soll beispielsweise ein Sensor per RS485 oder ähnlichen Feldbussen in das Netzwerk eingebunden werden, soll das Gerät ebenso per REST angesteuert werden können. Die zuvor geschaltete Adapter-Box dient dem Gerät demnach als Gateway-Interface zum Netzwerk bzw. der Management-Konsole.

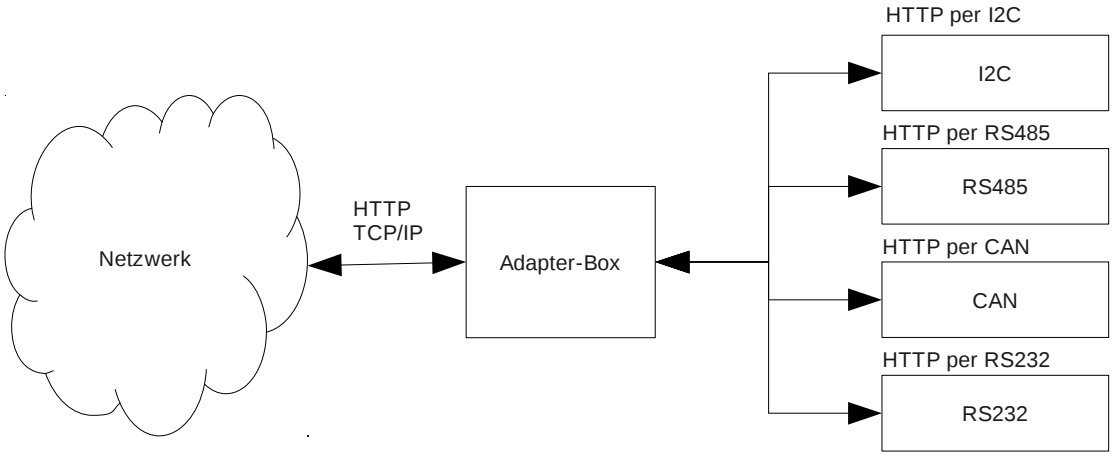


Abbildung 14: HTTP und die Kommunikationsstrecken

Für das Protokoll, das hinter der Adapter-Box verwendet wird, muss die HTTP-Nachricht in das passende Format gekapselt werden. Im Durchschnitt ist eine HTTP-Nachricht für das Netzwerk für Geräteansteuerungen nur einige 100 Byte groß. Für typische Embedded-Schnittstellen sind die Datenraten daher ausreichend.

In Abbildung 15 ist eine Integration in eine I2C-Nachricht dargestellt. Die Slave-Adresse kennt nur die Adapter-Box. Das I2C-Gerät kann dennoch von jedem anderen Teilnehmer über das interne Label adressiert werden. Kommunikationspartner müssen nur die Adapter-Box als Gateway-Interface benutzen, was aber transparent geschieht. IP-Adressen werden mittels Label an der Management-Konsole angefordert – für das Gerät wird die IP-Adresse des entsprechenden Gateways angegeben.

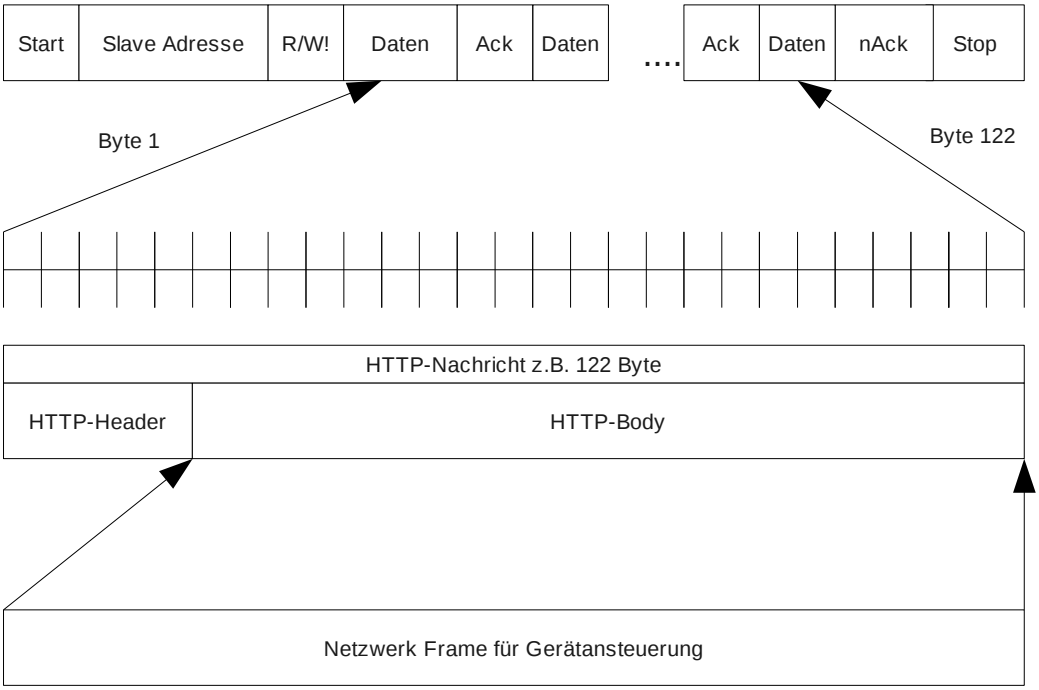


Abbildung 15: Integration HTTP-Nachricht in I2C-Nachricht

Auf die gleiche Art und Weise können die REST-Kommandos bei vielen anderen Protokollen und Schnittstellen integriert werden.

3.4. Infrastruktur / Caching / Skalierbarkeit

An dieser Stelle sei kurz das Thema Caching und Skalierbarkeit erwähnt. Für HTTP existieren viele für Inhalte von Webseiten geschriebene Anwendungen.

Es geht also um Proxyserver, Level-7 Switche, Load-Balancer, Beschleunigungsmodule wie FastCGI, etc.. In Webanwendungen können mit diesen Programmen „Service Level Agreements“- und „Quality of Service QoS“-Vorgaben (Kapitel 1.2. Datenkommunikation für Geschäftsprozesse) erreicht werden. Anwendungen für HTTP arbeiten stets auf den HTTP-Kommandos. Das bedeutet für ein GET auf ein Proxy auf eine URI, dass, wenn sich der Inhalt im Speicher befindet, dieser ausgeliefert werden soll, anstelle eine neue Anfrage an den eigentlichen Server zu senden.

Anwendungen dieser Softwaregruppe können auch für das hier entwickelte Netzwerk beim Einhalten der SLA- und QoS-Vorgaben unterstützen, es muss nur darauf geachtet werden, dass die Konfigurationen auf die Eigenschaften der Geräte abgestimmt werden.

3.5. Sicherheit

Über das Netzwerk werden kritische bzw. schützenswerte Mess-, Steuer und Regelungsdaten versendet. In Warenwirtschaftssystemen könnte man auf interne Zahlen schließen (Spionage), Material oder Produkte unbemerkt auslagern (Manipulation) oder in Steuerungen kritischer Geräte eingreifen. Zu jedem Zeitpunkt muss gesichert sein, dass nur die Anwendung des Netzwerks Kommandos versenden kann und darf.

Mögliche Angriffspunkte über das Netzwerk sind:

- Man-In-The-Middle Attacken
- Direkter Zugriff auf Geräte
- Replay Attacken
- Spionage / Sniffing / Logging

Für REST-Anwendungen existieren verschiedene Techniken für die Absicherung der Kommunikation. Passend für die Integration in ein Embedded-System muss ein Verfahren, das genügend Sicherheit bei vorhandener Rechenleistung gewährleistet, gewählt werden.

HTTP Basic Authentification

Dieses Verfahren dient als einfaches Authentifizierungsschema für Clients. Das Passwort geht unverschlüsselt (nur Base64-Codiert) an den Server. Jeder der das Passwort abgreifen kann, kann auf die Anwendung zugreifen.

HTTP Basic und HTTPS

Die Daten werden über eine verschlüsselte HTTPS-Verbindung transportiert. Ein abgreifen des Passworts ist durch die Verschlüsslung der Kommunikation nicht mehr möglich. Bei diesem Verfahren müssen beide Seiten SSL-Verschlüsselungsalgorithmen (asymmetrische und symmetrische) besitzen. Asymmetrische Verfahren benötigen sehr viel Rechenleistung, welche bei kleinen Mikroprozessoren nicht vorhanden ist.

HTTP Digest Authentification

Digest basiert auf dem Basic-Verfahren und möchte die Schwachstelle des unverschlüsselten Passwortes bei der Übertragung beseitigen. Realisiert wird dies durch den Einsatz der Hashfunktionen MD5 oder SHA. Mit den Ergebnissen dieser Funktionen kann nicht auf den Klartext geschlossen werden und zudem ergibt jeder Klartext einen anderen Wert. Zu beginn sendet der Server dem Client einen sogenannten „nonce“. Dies ist ein Hashwert. Der Client kann mit einem eigenem Passwort einen String wie folgt aufbauen [12]:

```
MD5(MD5(Benutzername | „:“ | Realm | „:“ | Passwort) | „:“ | nonce
| „:“ MD5(HTTP-Methode | „:“ | URI))
```

Der Server kann genau das gleiche Verfahren mit Kenntnis des Passworts nachrechnen und feststellen, ob die Authentifizierung erfolgreich ist oder nicht.

Digest erhöht die Sicherheit drastisch, denn das Passwort kann nicht mehr mitgelesen werden. Durch den Challenge-Response Prozess mit der „nonce“ ist eine Replay-Attache ebenfalls nicht mehr möglich.

XML-Encrypt / Cookie / HMAC / OpenID / OAuth

Viele weitere Verfahren existieren in diesem Bereich. Eine detaillierte Beschreibung kann hier [12] entnommen werden.

HTTP-Authentifikation für Adapter-Box und Management-Konsole

Aufgrund der geringen Rechenleistung und der aktuell noch nicht existierenden Bibliothek für eine SSL-Verbindung für den ausgewählten Mikroprozessor wird das Verfahren HTTP-Digest eingesetzt. Das Verfahren wird durch eine Nachrichtenverschlüsselung bzw. Prüfsumme auf der Nachricht erweitert.

```
MD5(MD5(Benutzername | „:“ | Realm | „:“ | Passwort | MD5(HTTP-
PARAMETER|DATA-FRAME)) | „:“ | nonce | „:“ MD5(HTTP-Methode | „:“ |
URI))
```

Die Integration der Prüfsumme der HTTP-Parameter und Frame-Daten schließt eine Man-In-The-Middle Attacke aus. Die Prüfsummen müssen entsprechend an den Softwarestack für die Verarbeitung der Kommandos weitergereicht werden.

Der reine HTTP-Body wird mittels AES verschlüsselt. Diese Nachrichtenverschlüsselung kombiniert mit den Hashwerten und Prüfsummen gewährleistet eine entsprechende Sicherheit. Steht eine SSL-Bibliothek für den eingesetzten Mikrocontroller zur Verfügung, sollte dieses Verfahren zum Einsatz kommen.

3.6. Asynchrone Verarbeitung

HTTP-Nachrichten werden typischerweise synchron verarbeitet (siehe Abbildung 16). Die Anfrage wird per TCP/IP an den Empfänger gesendet und die Antwort auf die Anfrage wird direkt im gleichen Kommunikationsfluss zurückgesendet.

Anfrage:

```
GET /infotext.html HTTP/1.1
Host: www.example.net
```

Antwort:

```
HTTP/1.1 200 OK
Server: Apache/1.3.29 (Unix) PHP/4.3.4
Content-Length: (Größe von infotext.html in Byte)
Content-Language: de (nach RFC 3282 sowie RFC 1766)
```

```
Content-Type: text/html
Connection: close

(Inhalt von infotext.html)
[17]
```

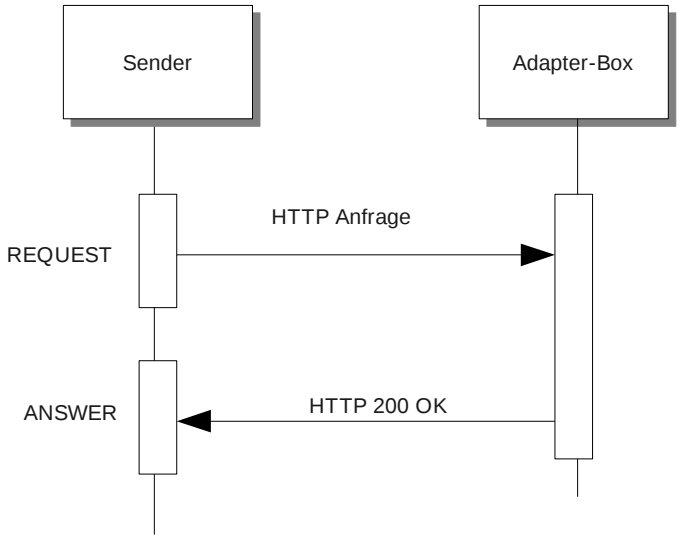


Abbildung 16: Synchrone Verarbeitung

In verteilten Systemen kann es sinnvoll sein, die Antwort asynchron zu senden, wenn die Verarbeitung der Anfrage vom Empfänger länger dauert. Dies bedeutet, dass der Sender dem Empfänger bei der Anfrage eine URL mitteilt, welche aufgerufen werden soll, wenn das Ergebnis zur Verfügung steht. In einer Programmiersprache wäre dies ein sogenannter typischer Callback-Aufruf. Optional könnte der Sender anstelle des Callback-Aufrufs eine URL des Empfängers zyklisch (Polling) abfragen, um festzustellen, ob ein Ergebnis bereit liegt. Abbildung 17 zeigt ein Ablaufdiagramm für einen asynchronen Ablauf.

Um eine Callback-Funktion bzw. Callback-URL aufrufen zu können, wird im Client zur Logik für die Auswertung und Entgegennahme von REST Zugriffen ein HTTP-Client benötigt, mit dessen Hilfe selbst Zugriffe auf andere Teilnehmer erzeugt werden können.

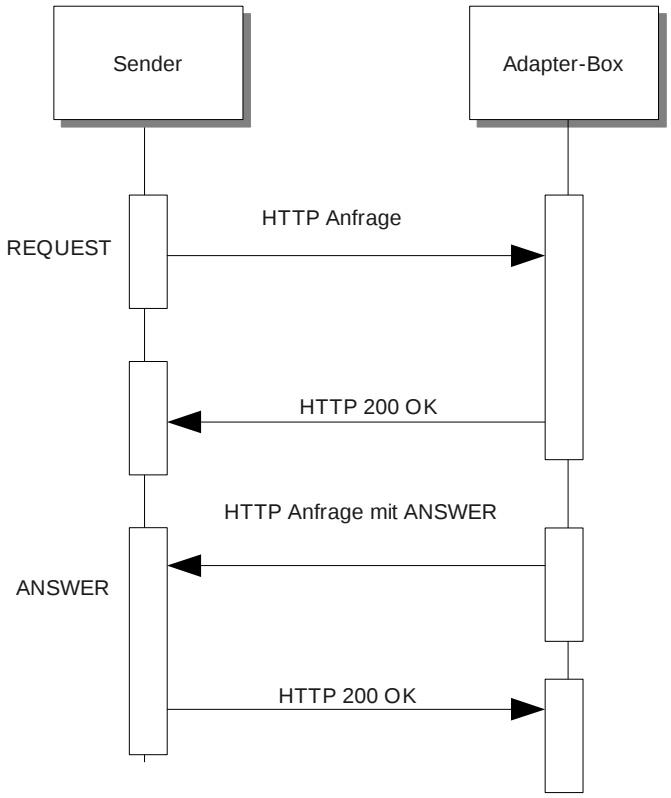


Abbildung 17: Asynchrone Verarbeitung

3.7. Ressourcen

Die Ressourcen in Tabelle 7 sind die Standardkommandos, die jede Adapter-Box erfüllen muss. Abhängig vom aktivierten Treiber kommen noch weitere Ressourcen hinzu.

Ressource	URI	Methode	Bemerkung
Uhrzeit	/clock	GET	
Uhrzeit setzen	/clock?value=UTC	POST	
Label abfragen	/abel	GET	
Label setzten	/abel?new={name}	GET	
IO Port abfragen	/gpio/get/{n}	GET	
IO Port setzen	/gpio/set/{n}	GET	
USB Bulk Transfer TX	/usb/bulk/write/{ep}?value={wert}	POST	
USB Bulk Transfer RX	/usb/bulk/read/{ep}	GET	
USB Contrrol Transfer	/usb/control?value={value}	GET,POST	
Debug LED blink	/ledblink	GET	Für Tests (3 mal blinken)
Info lesen	/info	GET	
Info setzten	/info?set={info}	GET	
SD Karten Inhalt lesen	/logflash	GET	
SD Karten Inhalt Marker setzte	/loglash?set={marker}	GET	Markierung setzen für bereits abgeholte Daten
Power	/power/off	GET	

Tabelle 7: Ressourcen Adapter-Box

Tabelle 8 ist die Liste der Ressourcen für die Management-Konsole. Im Laufe weiterer Entwicklungen wird die Liste noch wachsen.

Ressource	URI	Methode
Gerät anmelden	/devices	PUT,POST
Gerät IP-Adresse erfragen	/devices/{label}/ip	GET
Geräteliste	/devices	GET
Gerät ausschalten	/devices/{label}/power? state=on|off	GET
Geräte Alive-Meldung	/device/{label}/alive	GET

Tabelle 8: Ressourcen Management-Konsole

3.1. Testwerkzeug CURL

Für den Zeitpunkt in der Entwicklungsphase, in der noch keine eigene fertige Software existiert, kann für den Aufruf von REST Instanzen auf das Werkzeug CURL zurückgegriffen werden. Im Folgenden werden die wichtigsten Befehle aufgeführt:

- -X [action]: Ermöglicht die Angabe des HTTP Kommandos wie z.B. GET, POST, PUT oder DELETE.

  Beispiel: curl -X DELETE http://localhost:8080/devices/1

- -d [parameter]: Setzt die Variablen, die mittels POST übertragen werden sollen. Bei der Angabe dieses Parameters wird automatisch für den vorhergehenden -X POST gesetzt.

  Beispiel: curl -d "device[label]=waage" http://localhost:8080/devices

- -H [header]: Mit Header kann angegeben werden, welches Format der Client akzeptiert. Dadurch kann zwischen verschiedenen Repräsentationsformaten ausgewählt werden, vorausgesetzt es werden mehrere angeboten.

  Beispiel: curl -H "Accept: text/xml" http://localhost:8080/devices

Die einzelnen Parameter können kombiniert werden. Um beispielsweise ein Label eines Teilnehmers zu ändern, wird folgendes Kommando abgesendet:

```
curl -H "Accept: text/device" -X PUT -d "device[label]=waage001" \
http://localhost:8080/devices/1
```

Die Waage wird mit einem neuen Label versehen und der Client erwartet die Antwort auf die Ausgabe des Kommandos im Frameformat.

4. Struktur und Aufbau der Adapter-Box

Die Adapter-Box dient für jedes Gerät als Schnittstelle zur Integration in das Netzwerk. Das Embedded System muss möglichst einfach und kostengünstig konzipiert sein. Der Prozessor braucht genügend Rechenleistung, um einen TCP/IP Netzwerkstack ausführen zu können und um parallel den Zugriff auf ein externes Gerät zu steuern.

Damit möglichst viele verschiedene Geräte an die Adapter-Box angeschlossen werden können, muss eine große Vielfalt an bekannten Mikrocontroller- und typischen PC-Schnittstellen vorhanden sein.

Liste für die Anforderungen an die Schaltung:

- Kleine Bauform
- Ethernet RJ45 LAN Netzwerkinterface
- USB-Host Schnittstelle für USB-Geräte
- RS232-Anschluss
- Mikrocontroller-Schnittstellen (I2C, SPI,CAN, RS485, etc.)
- Optimiert für Low-Power (Standby)
- „Power over Ethernet“ Versorgungsspannung
- Steckernetzteil Anschluss
- Schaltbarer 230V Anschluss
- Diverse IO- und AD-Ports (Digitale Ein- und Ausgänge, Analog-Digital-Wandler)

Eine Übersicht der Komponenten ist in Abbildung 18 zu sehen.

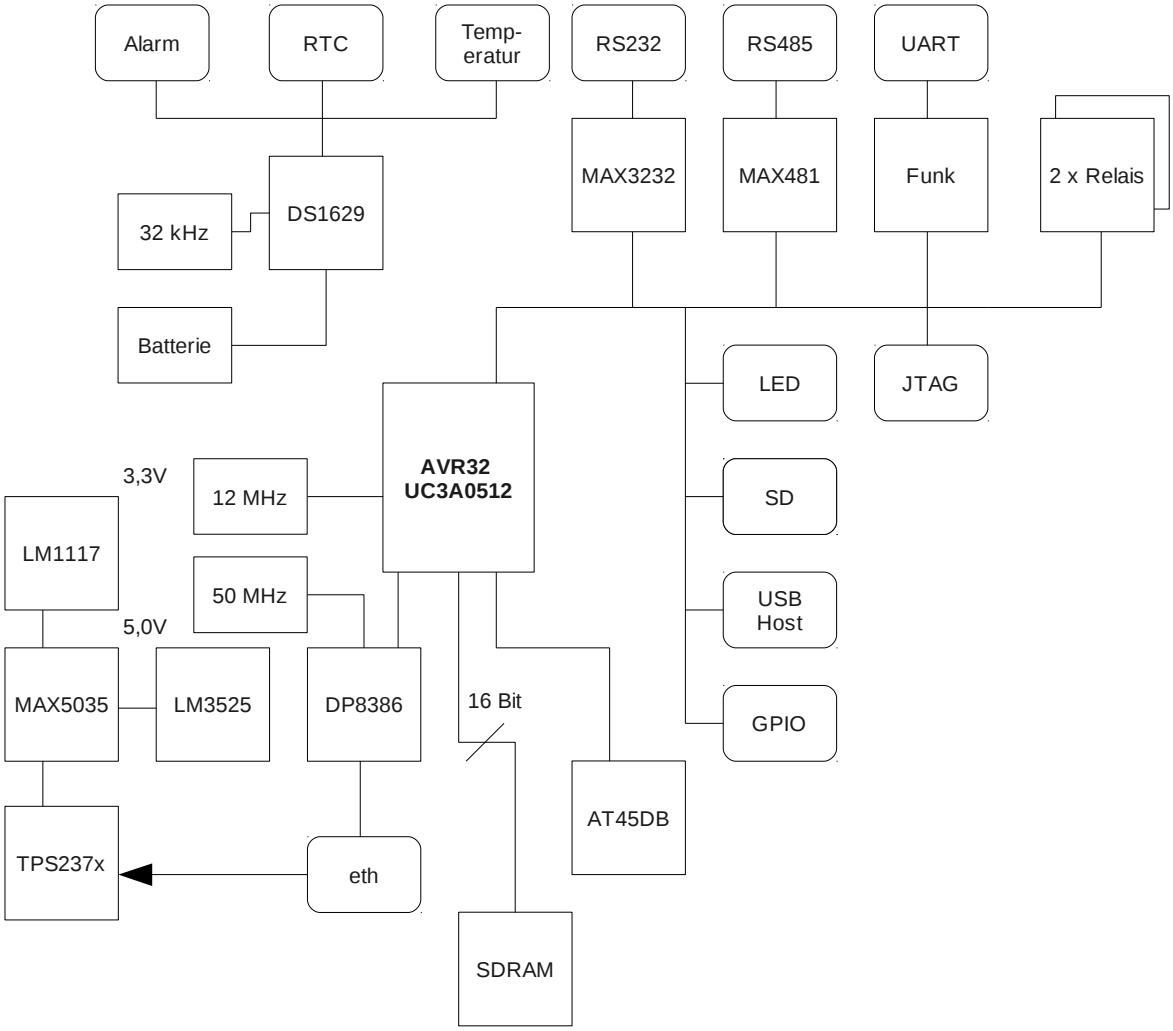


Abbildung 18: Übersicht Adapter-Box

In diesem Kapitel wird auf die technischen Implementierung der Schaltung und die Konzeption bzw. Auswahl der Bauelemente eingegangen. Alle Schaltpläne, Layoutdaten, Stücklisten und Produktionsunterlagen (siehe CD-ROM) sind im Rahmen der Masterarbeit entstanden und aufgebaut worden.

4.1. Platzierung / Bestückung

Der Aufbau der Platine ist wie in Abbildung 19 dargestellt definiert worden. Auf der linken Seite befinden sich alle Schnittstellen für die Ansteuerung der Peripherie: Eine USB-Buchse für USB-Geräte wie Etiketten-Drucker, Barcode-Scanner, etc. und ein 20-poliger Multifunktionsstecker, auf welchem viele bekannte Mikrocontroller-Schnittstellen, IO-Ports

und AD-Kanäle abgegriffen werden können. Zwei LEDs ermöglichen eine optische Ausgabe über den Betriebszustand.

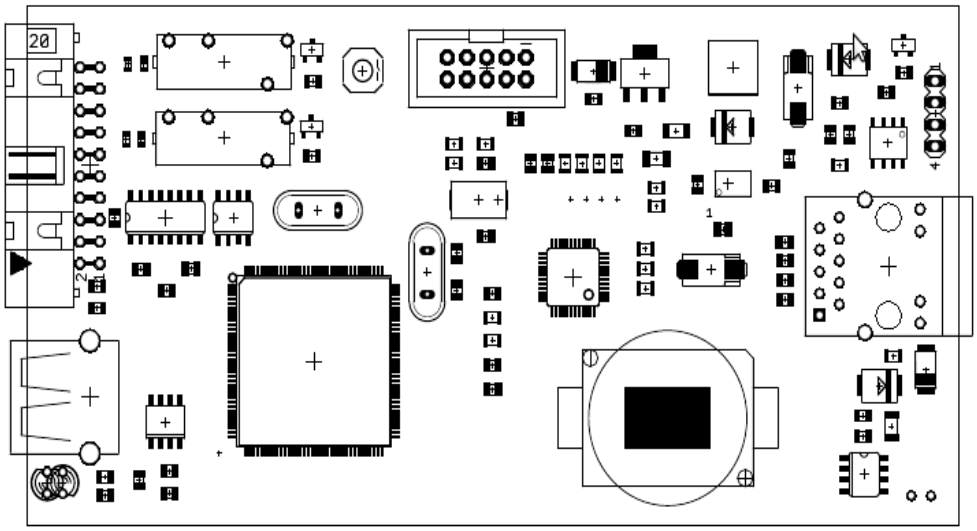


Abbildung 19: Bestückung

Auf der rechten Seite befinden sich eine Netzwerkbuchse (RJ45) für die Ansteuerung über das Netzwerk. In einem weiteren Designdurchlauf der Schaltung soll die benötigte Fläche nochmals halbiert werden.

4.2. Prozessor

Wie bereits beschrieben, sind die Auswahlkriterien für den Prozessor die unterstütze LAN-Schnittstelle bzw. genug Rechenleistung für einen Netzwerkstack, um die REST-Kommunikation zu ermöglichen, eine USB-Schnittstelle für externe USB-Geräte und eine möglichst große Vielfalt an typischen Mikrocontroller-Schnittstellen.

Als Mikrocontroller kommt ein AT32UC3A0512 von Atmel zum Einsatz. Er bietet eine 32-bit AVR UC Architektur an, die für kleine mobile Geräte optimiert ist. Im Prozessor sind 512 KB Flash, 64 KB SRAM, 10/100 Ethernet MAC und eine OTG USB 2.0 Schnittstelle integriert. Der Prozessor besitzt zudem ein externes Speicherinterface für die Verwendung eines externen SDRAM-Speichers. Für die Prozessoren gibt es eine Portierung des freien

GNU/GCC Compilers. Der Prozessor bietet zwei externe 32-Bit Ports mit - über eine interne Matrix konfigurierbare - IO-Pins.

Die große 100-Pin Version im TQFP-100 Gehäuse bietet zusätzlich noch einen Port an, an dem ein externer Speicherbaustein angeschlossen werden kann. Die 3,3 Volt Versorgungsspannung wird entsprechend an den Versorgungspin angeschlossen. Ein interner 1,8V Spannungsregler erzeugt für den Kern die notwendige Spannung, die ebenfalls außen am Chip auf die entsprechenden Versorgungsleitungen gelegt werden muss.

Der Prozessorkern benötigt eine externe Taktquelle und eine entsprechende Resetschaltung, wie in Abbildung 20 und 21 angeschlossen. Zu den prozessornahen Pins gehören ebenfalls die JTAG Leitungen TDI, TDO, TMS und TCK, welche eine externe Programmierung des internen Flash und In-Circuit Debugging bzw. einen Boundary-Scan ermöglichen.

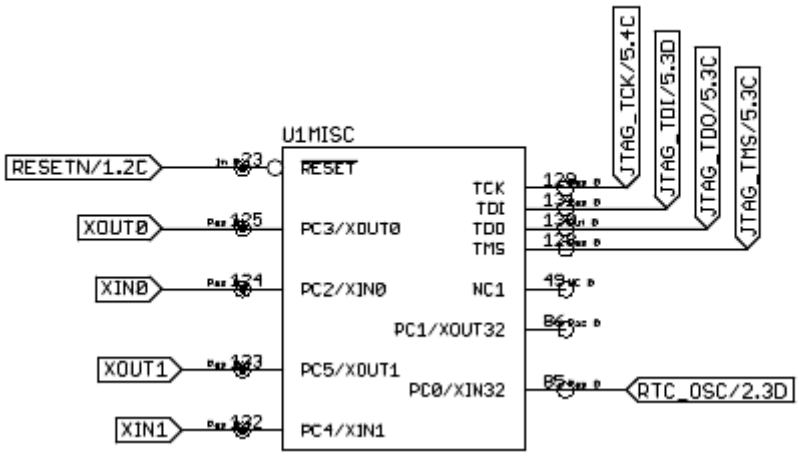


Abbildung 20: Takt und JTAG

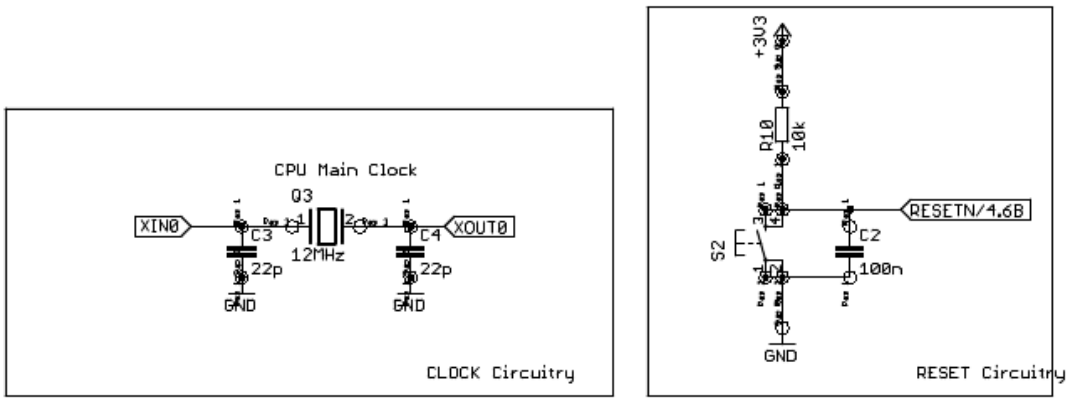


Abbildung 21: Quarz- und Reset-Schaltung

4.3. Externer SDRAM-Baustein

Als externer Speicher wurde ein Standard SDRAM-Baustein (siehe Abbildung 22) gewählt. In der aktuellen Bestückung befindet sich ein 64MBit großer Speicher für das Betriebssystem und zum Empfang von Netzwerkdaten. Abhängig von den Treibern für die externen Geräte kann zur Einsparung von Kosten auf den zusätzlichen Speicher verzichtet werden.

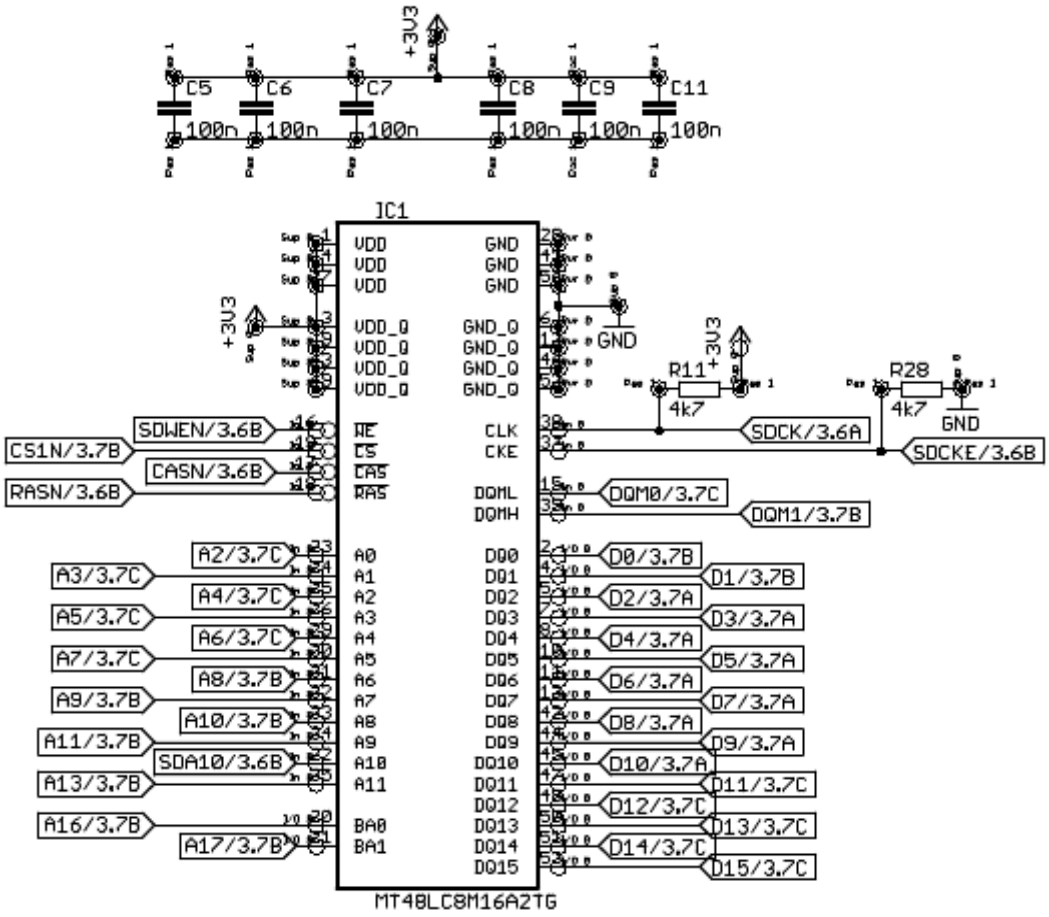


Abbildung 22: Externer SDRAM Speicherbaustein

4.4. Power-Management

Die Adapter-Box kann in zwei Betriebszustände geschaltet werden. Den Standby-Modus, in welchem der Prozessor „schläft“ und über externe Ereignisse geweckt werden kann, und den aktiven Betriebszustand, in welchem alle Funktionen über das Netzwerk zur Verfügung stehen. Mit Strom kann die Adapter-Box entweder über ein Netzteil am 20-poligen Multifunktionsstecker oder über ein Power over Ethernet Switch versorgt werden. Die

Schaltung der Adapter-Box benötigt im Standby maximal 100 µA. Aufgeweckt werden kann die Schaltung über ein WoL Signal an den Prozessor, einen zuvor programmierten internen RTC-Alarm, ein Ereignis an einem IO-Port oder über einen zyklischen Polling-Mechanismus, bei dem jede Adapter-Box regelmäßig bei der Management-Konsole nachfragt, ob jemand nach ihr gefragt hat.

	Extern	**Intern**
Wake on LAN (Magic Paket)	LAN-Schnittstelle	NET_IRQ
Externen Schalter steuern	IO-Ports	Eingang IO1 als IRQ (Wakeup)
Zeitgesteuert	Programmierung zuvor	RTC ALARM
Funk	Polliing zur Management-Konsole	

Tabelle 9: Ein- und Ausschaltmöglichkeiten für Anwendung und Adapter-Box

Power over Ethernet

Durch den Einsatz der Technologie „Power over Ethernet“ (gemäß IEEE 802.3af-Standard), in der eine integrierte Stromversorgung über das LAN-Kabel für die Versorgung des Geräts genutzt werden kann (die Technik wurde vor allem für WLAN Router, VoIP Telefon, Kameras, usw.. entwickelt.), kann eine zentrale Stromversorgung kostengünstig erreicht werden. Abbildung 23 zeigt für die Masterarbeit verwendeten Switch der Firma Netgear, welcher auf den ersten vier Ports eine integrierte Stromversorgung mitliefert.

Abbildung 23: Netgear Power over Ethernet Switch

Mithilfe des Bausteins TPS237x (Abbildung 24) der Firma Texas-Instruments kann die angelegte Wechselspannung auf dem LAN-Kabel zurückgewonnen werden. Der Standard bietet hierfür verschiedene Leistungsklassen an (Tabelle 11). An den Ausgängen POE+ und POE- liegen im aktiven Zustand ca. 30-39 V Spannung an. Bei der Aktivierung des maximal möglichen Stroms, der über einen Port des Switch geliefert werden kann, muss eine genaue Einschaltphase eingehalten werden, in der nacheinander [19] verschiedene Widerstände angeschlossen werden ,um in die verschiedenen Zustände (Abbildung 25 und Tabelle 10) wechseln zu können.

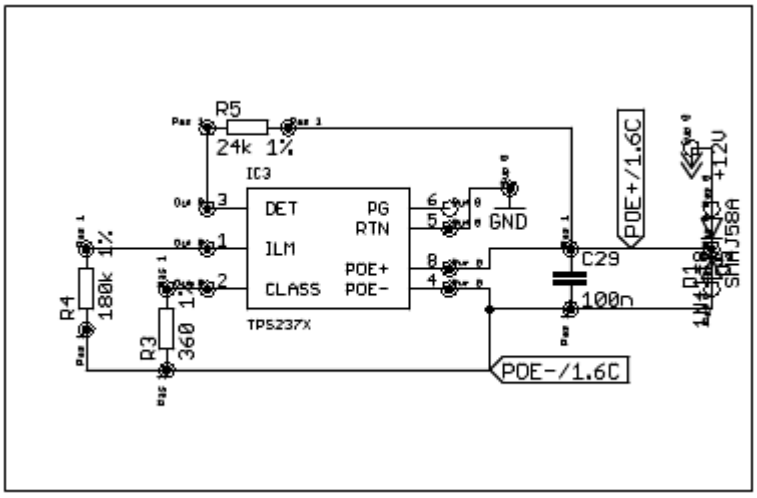


Abbildung 24: Power over Ethernet

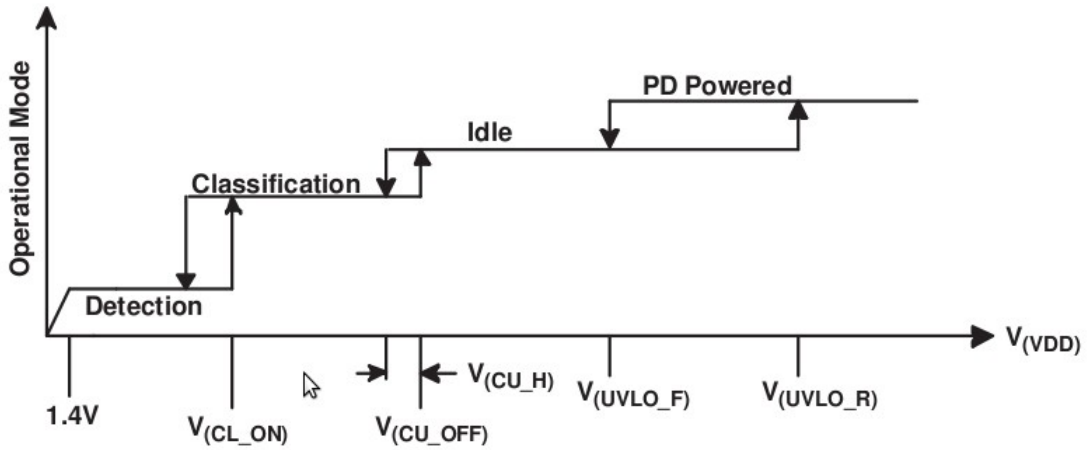


Abbildung 25: TPS237x [18]

Schritt	Aktion	Spannungsbereich
Detektion	Feststellung ob Endgerät einen Widerstand im Bereich von 15–33 kΩ aufweist	2,7–10,0
Klassifikation	Messung des genauen Widerstandswertes um Leistungsklasse festzustellen.	14,5–20,5
Startup	Eigentliche Stromversorgung aktivieren	>42
Normale Operation	Stromversorgung im Versorgungsmodus	36–57

Tabelle 10: Aktivierung bei Power over Ethernet

Klasse	Verfügbare Leistung in Watt am versorgten Gerät
0	0,44–12,96
1	0,44–3,84
2	3,84–6,49
3	6,49–12,95
4	Reserviert, referenziert Klasse 0

Tabelle 11: Verfügbare Leistungsklassen

Wake on LAN (Magic Paket)

Die von AMD und Hewlett Packard 1995 erfundene Technik, ein System über die LAN-Schnittstelle zu wecken hat mittlerweile auch Einzug in die Mikrocontrollertechnik gefunden. Mittels dem sogenannten „Magic Packet“, ARP-Request oder ARP abhängig von der IP-Adresse kann ein AT32UC3A0512 Prozessor den Betriebszustand wechseln. Abbildung 26 zeigt das interne Register für die Konfiguration.

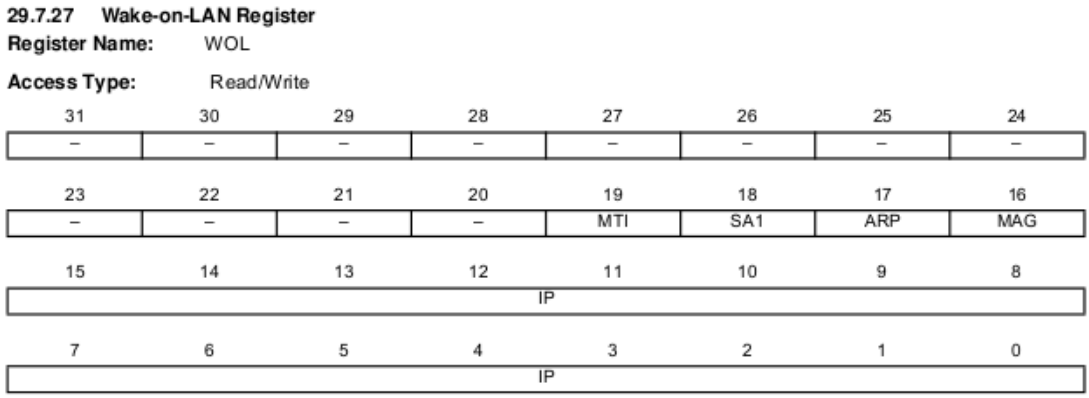


Abbildung 26: Wake on LAN Register UC3A

• IP: ARP request IP address

Written to define the least significant 16 bits of the target IP address that is matched to generate a Wake-on-LAN event. A value of zero does not generate an event, even if this is matched by the received frame.

• MAG: Magic packet event enable

When set, magic packet events causes the wol output to be asserted.

• ARP: ARP request event enable

When set, ARP request events causes the wol output to be asserted.

• SA1: Specific address register 1 event enable

When set, specific address 1 events causes the wol output to be asserted.

• **MTI: Multicast hash event enable**

When set, multicast hash events causes the wol output to be asserted.

IO-Pin als Einschalter

Mit einem Ereignis auf einem externen IO-Pin kann die Schaltung ebenfalls aufgeweckt werden. Das Signal kann entweder durch einen Sensor oder einen externen Taster ausgelöst werden.

Zeitgesteuert

In der Schaltung befindet sich ein eigens Batteriegepufferter Zeitbaustein DS1629 von Maxim-IC. In den Baustein können Zeitpunkte programmiert werden, an denen ein Alarmsignal ausgegeben wird. Das Alarmsignal wiederum ist mit einer externen Interruptleitung des AT32UC3 verbunden, welcher ebenfalls im schlafenden Zustand reagiert (siehe Abbildung 27).

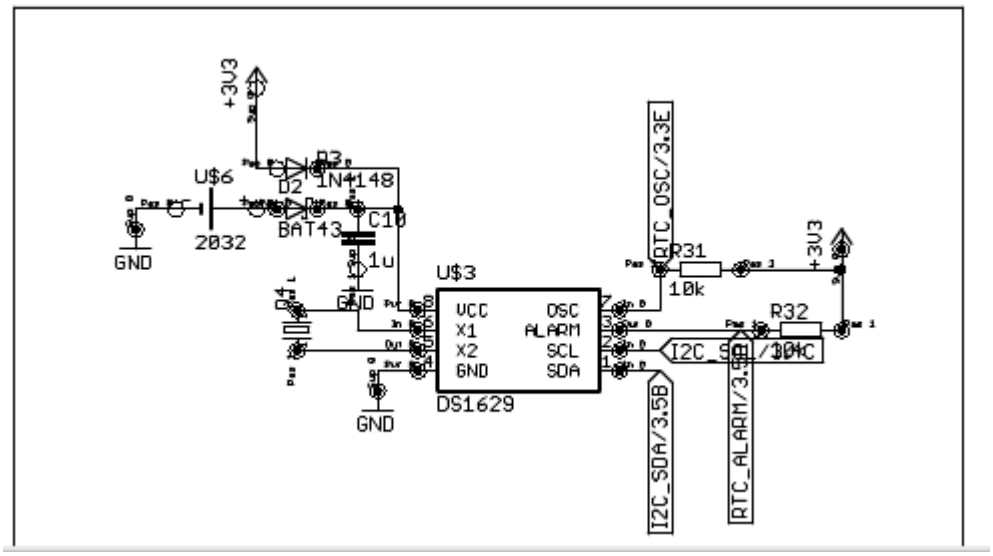


Abbildung 27: Realzeituhr mit Alarmfunktion

Optionales Funknetzwerk

Auf der Schaltung befindet sich ein optionaler Anschluss für ein Funkmodul. Dadurch kann die Adapter-Box in einer späteren Version erweitert werden.

Anwendungsideen:

- Internes Fall-Back Netzwerk
- Integration von Geräten per Funk in das Netzwerk
- Tragbare mit Batterie versorgte Adapter-Box (LCD und Taster für Rückmeldungen)

Durch regelmäßiges Polling - initiiert durch den Prozessor - kann die Adapter-Box an einem anderen Funknetzwerk-Teilnehmer bei der Management-Konsole nachfragen, ob Bedarf an dem eigenen Gerät besteht und entsprechend reagieren.

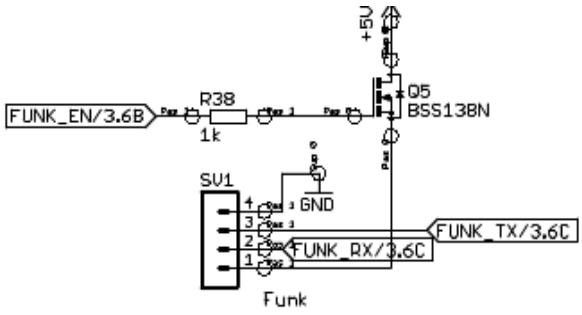


Abbildung 28: Optionaler Anschluss für Funkschaltung

Als optionale Funkschnittstelle ist aktuell das Produkt „ICradio Mini 2.4G“ der Firma In-Circuit GmbH (Abbildung 29) angedacht. Das Modul bietet über eine UART-Schnittstelle die Funkstrecke als einfach anzusteuernden Kanal an. In Abbildung 28 ist der Anschluss auf der Adapter-Box gezeigt. Mit einem externen FET kann die Stromversorgung durch den AT32UC3 entsprechend ein- und ausgeschaltet werden. Dadurch ist ein Ressourcen schonender Zugriff auf das Funkmodul gewährleistet, denn der extra Prozessor auf dem Funkmodul samt Ansteuerung der Antenne benötigt nur ca. 30 mA an Strom. In einem

weiteren Redesign der Schaltung soll das Funkmodul selbst aufgelöst auf die eigene Schaltung kommen.

Abbildung 29: ICradio Mini 2.4G

Speicher für Logdateien

Lokale Ereignisse und Zugriffe werden durch die Adapter-Box in einem nicht flüchtigen Speicher mitprotokolliert. Über die Management-Konsole können diese Speicher regelmäßig in eine zentrale Datenbank geschrieben werden. In der ersten Version der Schaltung (welche der Masterarbeit zugrunde liegt) befindet sich noch ein Data-Flash Baustein AT45DB (Abbildung 30) auf der Schaltung.

Aus mehreren praktischen Gründen jedoch wird dieser in der nächsten Version entfernt, und es wir ausschließlich der Einsatz einer SD-Karte (Abbildung 31) unterstützt. SD-Karten mit Kapazitäten von 1-2GB sind für die Anwendung völlig ausreichend und kosten aktuell nur noch ca. 1-2 EUR. Die SD-Karte kann mittels Leser ebenfalls an einen PC angeschlossen werden. Logdateien könnten so auch einfach kopiert werden. Desweiteren können Firmare-Upgrades ebenfalls über die SD-Karte in die Adapter-Box eingespielt werden.

Mittels Webservice-Aufruf kann so bequem ein Firmware-Update erst auf die SD-Karte geschrieben und nachts entsprechend durchgeführt werden.

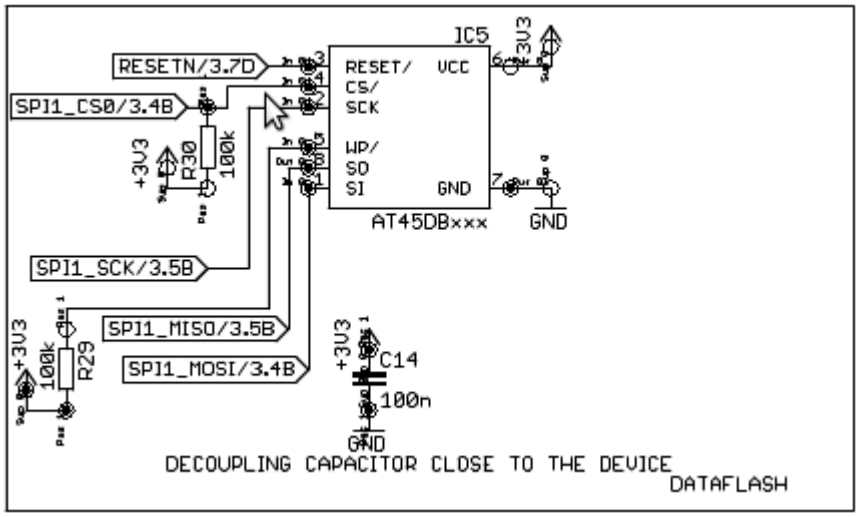


Abbildung 30: Dataflash Speicher

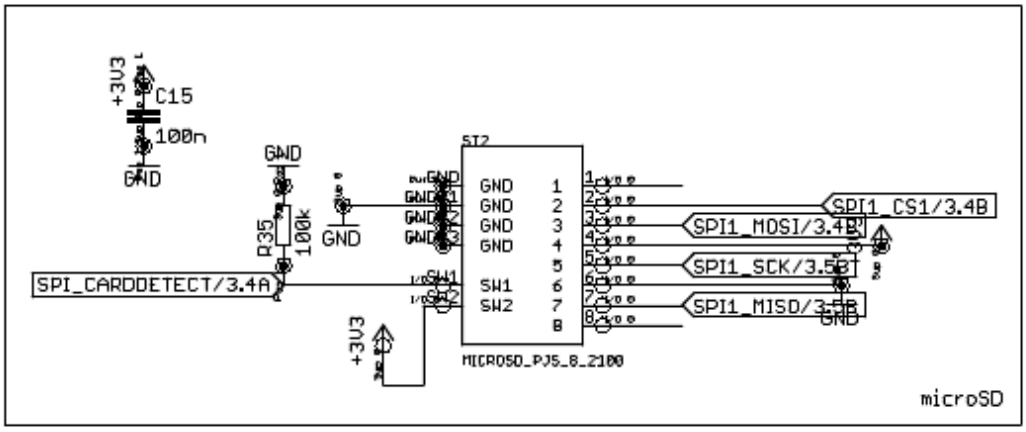


Abbildung 31: SD-Karten Halterung

Statusanzeige

Für die Anzeige des Betriebszustands bzw. für Fehlverhalten existieren zwei LEDs, die am Gehäuse nach außen geführt sind. Die LEDs können entweder intern durch die Firmware oder extern per Webservice angesteuert werden (siehe Abbildung 32).

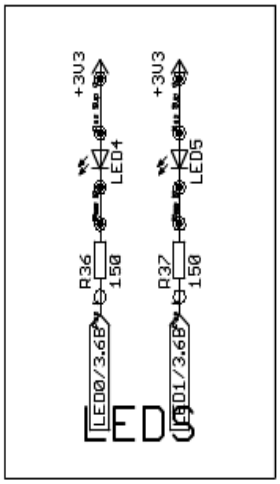


Abbildung 32: Statusleuchtdioden

4.5. Stromversorgung

Die Stromversorgung versorgt nicht nur die Adapter-Schaltung, auch extern angeschlossene USB-Geräte müssen versorgt werden. Es wird daher eine dementsprechend ausgelegte Stromversorgung zur Verfügung stehen. Der Baustein MAX5035 (siehe Abbildung 33) bietet eine Leistung von bis zu 1A bei einer Eingangsspannung von bis zu 76 V an. Die hohe Eingangsspannung ist sehr wichtig, denn der „Power over Ethernet“ Controller liefert eine Spannung von ca. 30-40V.

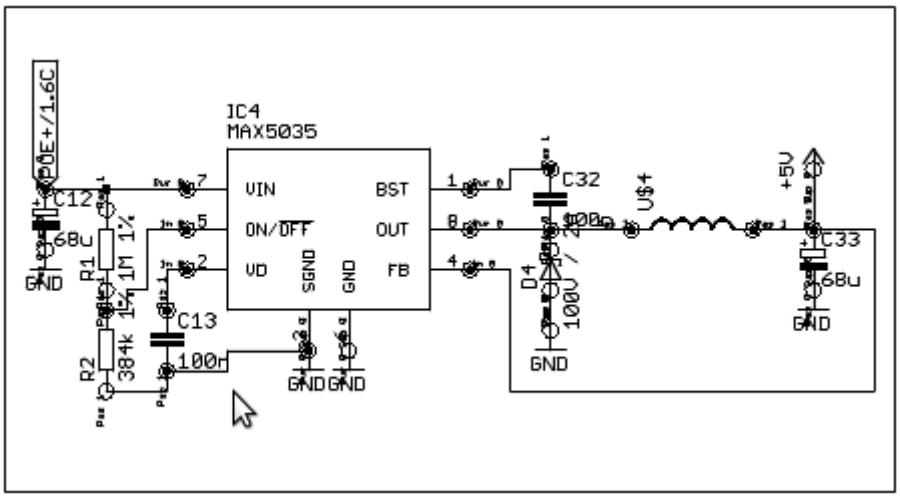


Abbildung 33: MAX5035

Aus der erzeugten 5V Spannung durch den getakteten Regler MAX5035 kann mittels einem einfachen LM1117 Linear Regler (Abbildung 34) die 3,3V Betriebsspannung für den Mikrocontroller und den Arbeitsspeicher erzeugt werden.

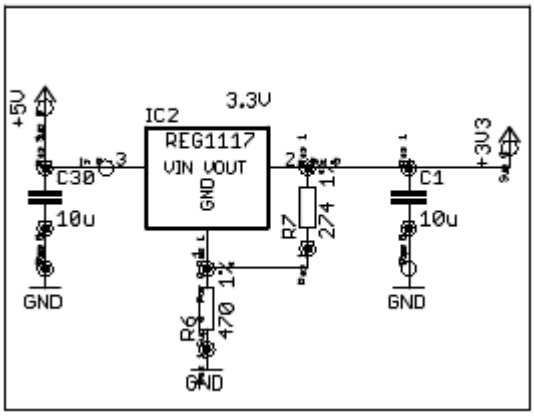


Abbildung 34: LM1117 3,3V Regler

Extern angeschlossene USB-Geräte müssen über die interne Stromversorgung angeschlossen werden. Laut USB-Spezifikation dürfen externe Geräte bis zu 500 mA von ihrem Hostrechner anfordern. Da die Adapter-Box in diesem Fall der Host ist, muss der entsprechende Strom zu Verfügung stehen. In der Adapter-Box werden externe Geräte

über einen extra Baustein mit namens LM3525 angeschlossen (siehe Abbildung 35). Der Baustein bietet folgende Optionen:

- Einfaches aktivieren und deaktivieren der Spannung (digitaler Eingang)
- Erkennung von Kurzschlüssen mit Schutzmechanismus (digitaler Ausgang)

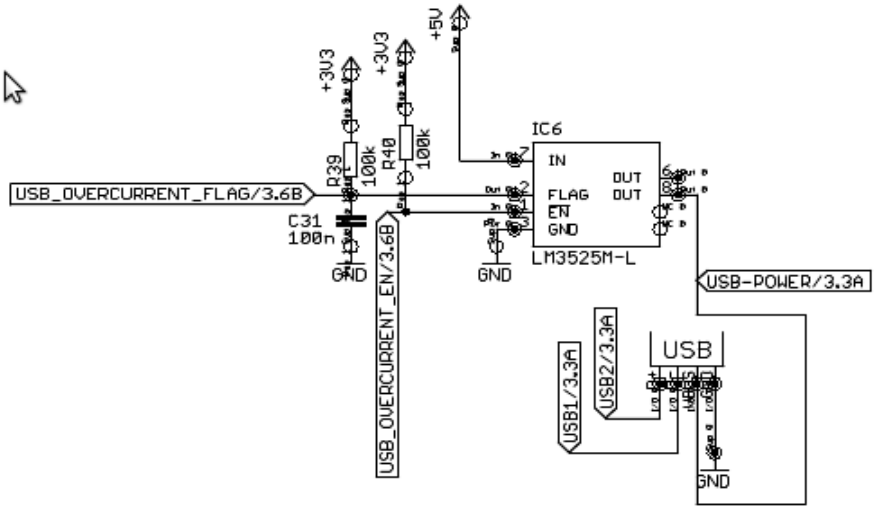


Abbildung 35: LM3525 USB-Spannungscontroller

Mit diesem Baustein, kann in der Firmware auch entschieden werden, wann das externe Gerät aktiviert oder deaktiviert wird (z.B. nach Verlassen des internen Sleep-Modes).

4.6. Schnittstellen

Im Folgenden werden kurz die Schnittstellen an der Adapter-Box aufgelistet und beschrieben.

Ethernet RJ45

Das Netzwerkinterface (Abbildung 36) dient zum Anschluss des Ethernet über ein RJ45 Kabel. Als Buchse wurde eine solche gewählt, welche die Leitungen für den Anschluss der „Power over Ethernet Technologie“ ermöglicht. Die RX und TX Leitungen gehen vom Stecker zum nächsten Baustein, den Netzwerk-PHY (Abbildung 37), der für die Aufbereitung der Bitströme auf dem Netzwerk zuständig ist. Für die PoE Stromversorgung werden die Leitungen POE+ und POE- zum Wandler TPS237x geschaltet.

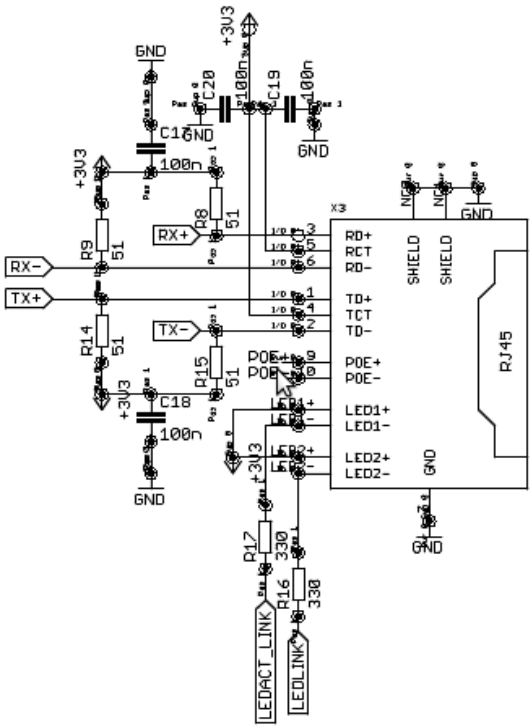


Abbildung 36: Netzwerkanschluss

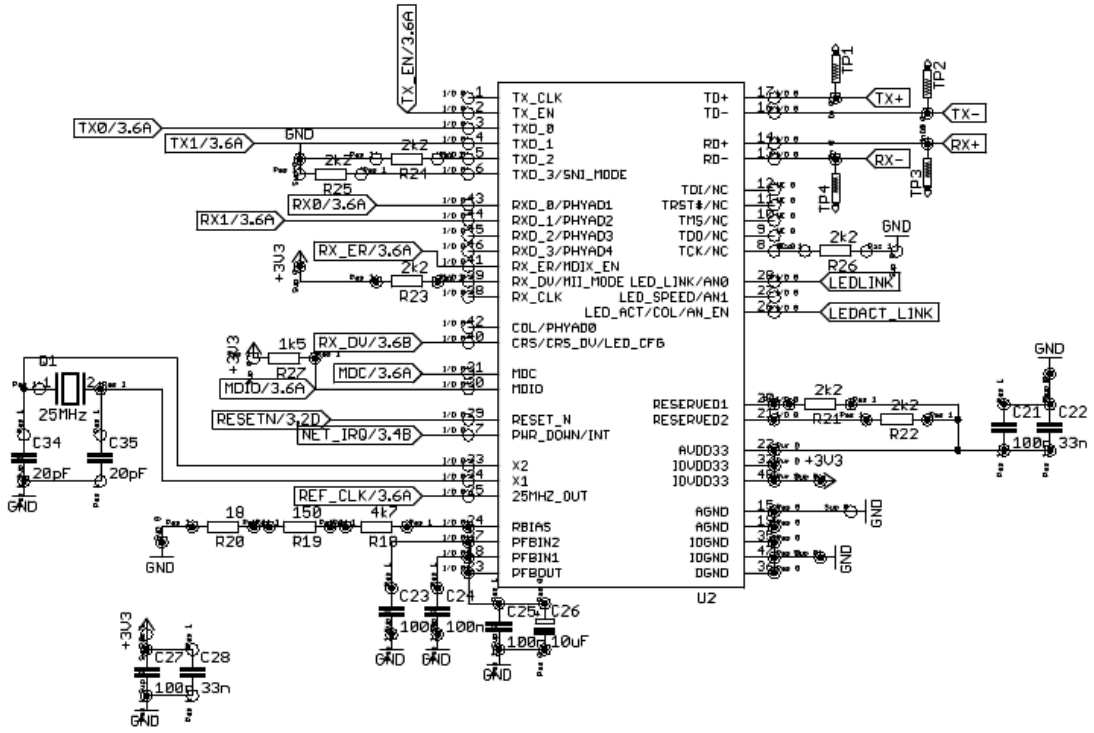


Abbildung 37: Netzwerk National Semiconductor DP8386

Die Beschaltung wurde Application-Notes, sowie dem Evaluationsboard EVK1100 von Atmel entnommen.

RS232-Schnittstelle

Auf dem Markt befinden sich trotz USB noch viele RS232 Geräte. Um diese direkt an die Adapter-Box anschließen zu können wurde ein UART-Transceiver (Abbildung 38) mit in die Schaltung aufgenommen. Der UART-Transceiver erzeugt aus den asynchronen Spannungen auf den TX- und RX-Leitungen 3,3V kompatible Pegel für den Mikrocontroller. Maximale Baudraten von bis zu 115200 Kbaud sind mit dieser Beschaltung möglich.

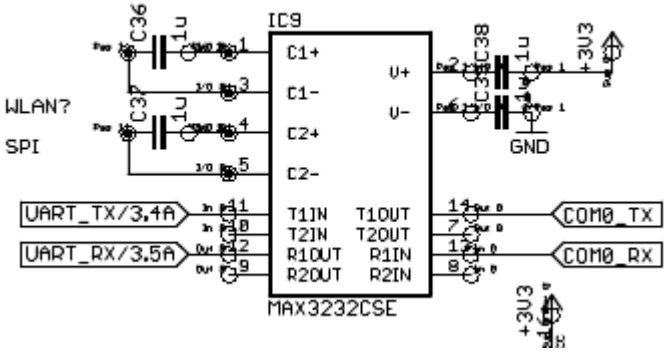


Abbildung 38: RS232 Anschluss

USB-Schnittstelle

Über eine USB-A-Buchse (Buchse wie an einem Computer) können USB-Geräte angesteuert werden. Der Anschluss von USB-Geräten bedarf keiner weiteren Bausteine. Die USB 2.0 Schnittstelle, die „Full-Speed"-fähig ist, kann vollständig in den AT32UC3A integriert werden. Ausgelegt ist die USB-Schnittstelle als OTG-Ausführung, d.h. die Schnittstelle kann als Host oder Device verwendet werden. In der Anwendung der Adapter-Box wird diese als Host eingesetzt, um so USB-Geräte ansteuern zu können.

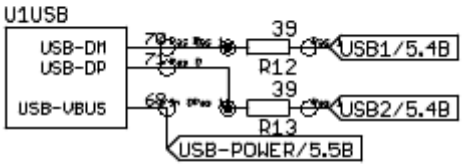


Abbildung 39: USB-Buchse

RS485-Schnittstelle

Als Vorbereitung für den späteren Einsatz von RS485 Geräten dient der Baustein MAX481CSA (Abbildung 40). Der RS485 Bus wird gerne bei Hausvernetzungen oder Steuerungen verwendet. Auf dem Markt gibt es daher viel Equipment wie Türöffner, Sensoren, Rolladensteuerungen, RFID-Kartenleser, Motoransteuerungen, Lichtsteuerungen, usw.

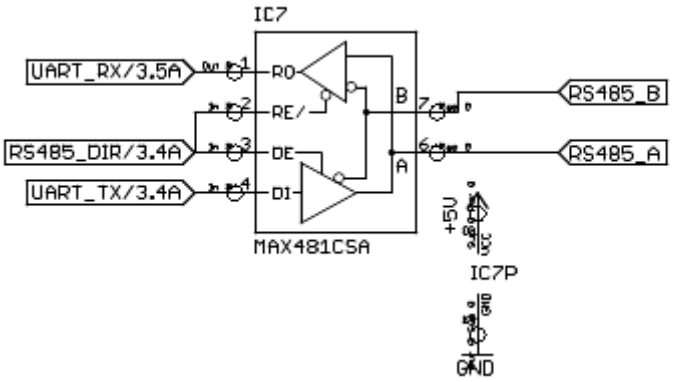


Abbildung 40: RS485 Transceiver

Der Prozessor UC3A kann einen UART in dem Betriebsmodus RS485 schalten. Im Rahmen der Masterarbeit wurde jedoch noch keine Softwareimplementierung vorgenommen.

Multifunktionsstecker

Um das Gehäuse klein zu halten, wurde eine 20-polige Anschlussleiste mit allen ein- und ausgehenden Signalen belegt. Später soll es für bestimmte Geräte vorkonfektionierte Kabel geben, um einen einfachen aber schnellen Aufbau zu gewährleisten. In Tabelle 12 sind alle Anschlüsse aufgelistet. Die beiden Relais auf der Schaltung (Abbildung 42) sind über die Anschlüsse 13, 15, 17 und 19 erreichbar.

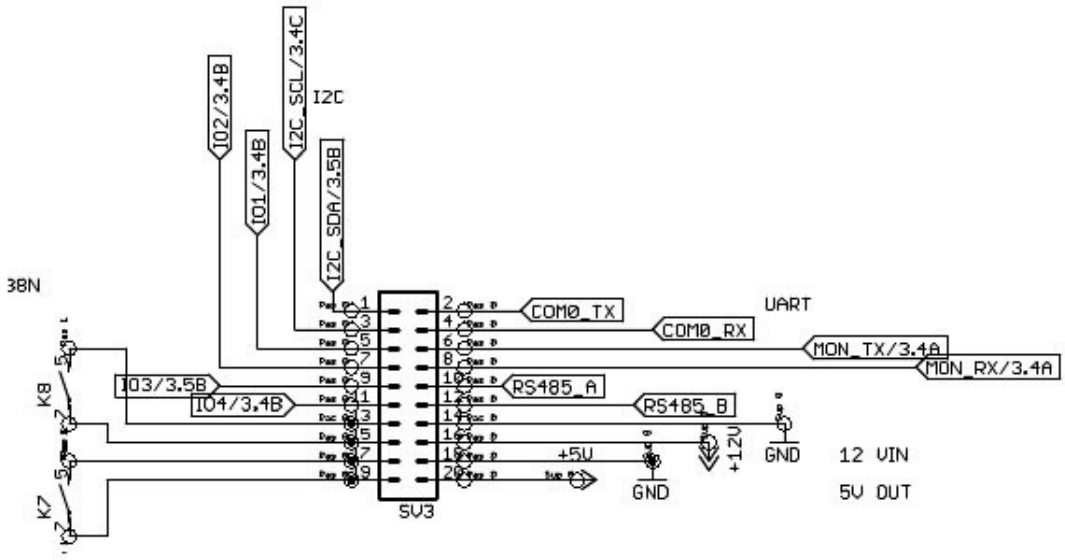


Abbildung 41: Multifunktionsstecker

Bezeichnung	Pin	Pin	Bezeichnung
I2C SDA	1	2	RS232 TX
I2C SCL	3	4	RS232 RX
IO1	5	6	Debug TX
IO2	7	8	Debug RX
IO3	9	10	RS485 A
IO4	11	12	RS485 B
Relais 1 Kontakt 1	13	14	GND
Relais 1 Kontakt 2	15	16	12 V Eingang
Relais 2 Kontakt 1	17	18	GND
Relais 2 Kontakt 2	19	20	5 V Ausgang

Tabelle 12: Pinbelegung Multifunktionsstecker

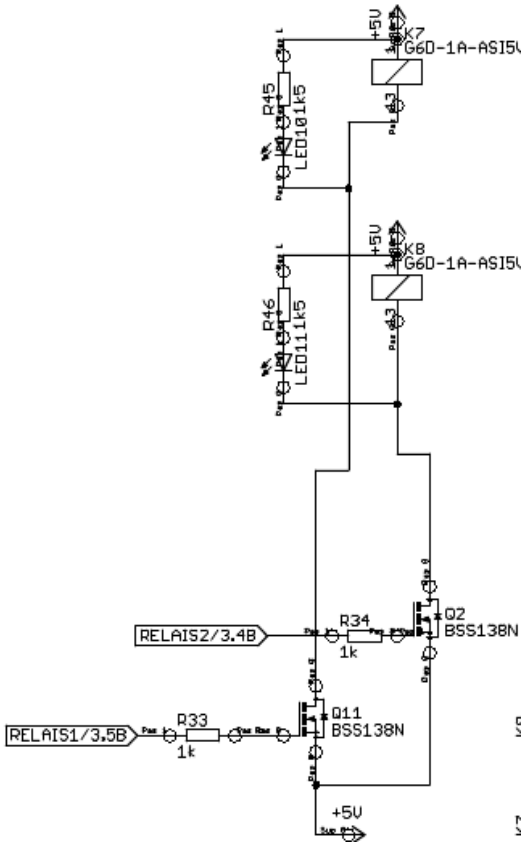


Abbildung 42: Relais

4.7. Softwarekomponenten

Aufgabe der Adapter-Box ist die Integration von externen Geräten als Webservice in das lokale Netzwerk mit Registrierung an der Management-Konsole. Wie bereits in vorherigen Kapiteln erwähnt, werden hierfür ein TCP/IP Netzwerkstack, HTTP-Client und die Treiberroutine für das angeschlossene Gerät (USB, RS232, o.ä.) benötigt. Für die Kommunikation kommen noch der REST-Server inkl. der kryptografischen Funktionen für die Authentifizierung und Nachrichtenverschlüsselung hinzu.

In Abbildung 43 ist ein Grobentwurf für den Ablauf des internen Prozesses der Adapter-Box abgebildet.

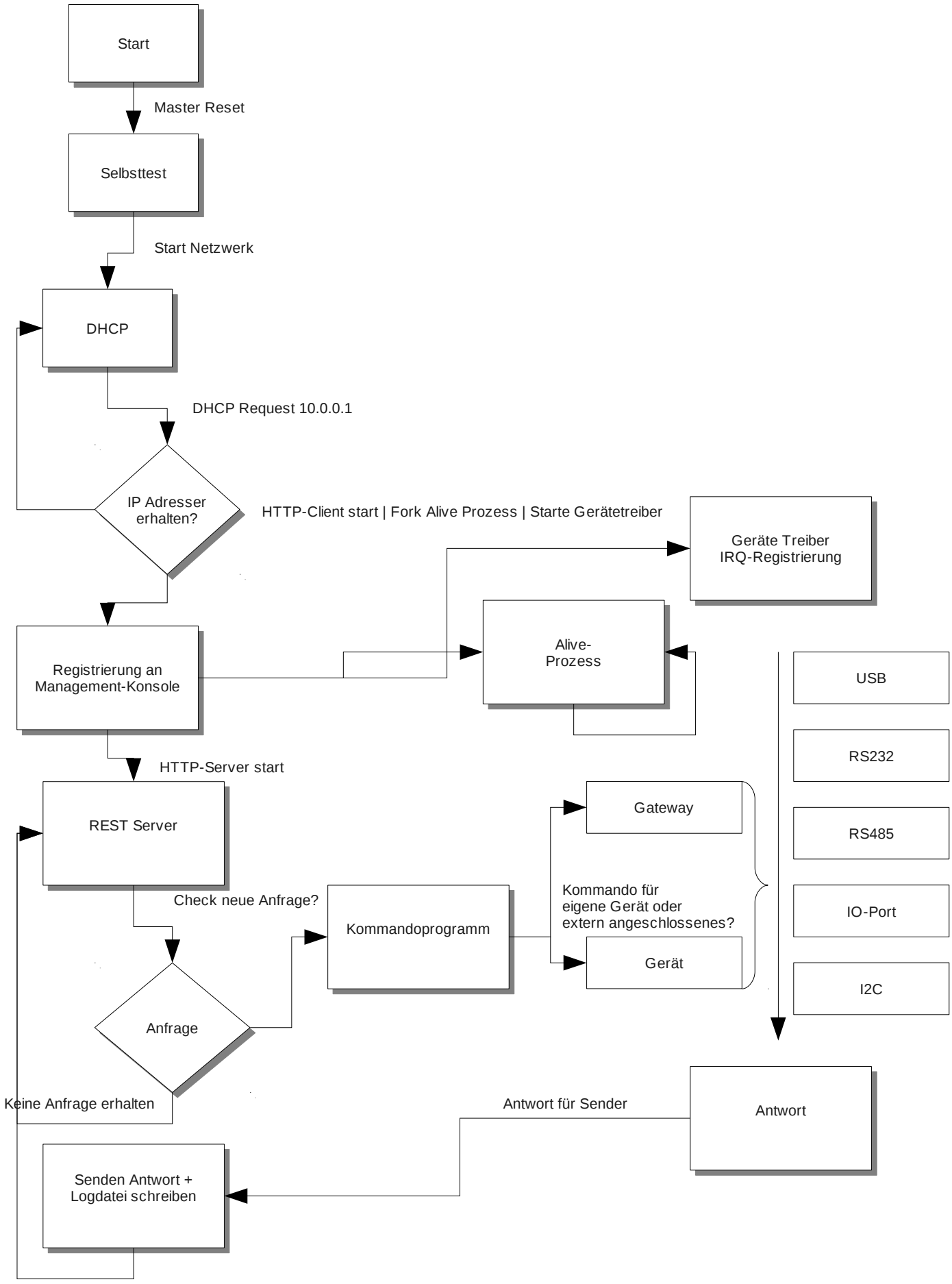


Abbildung 43: Software Adapter-Box

Der Ablauf startet mit einem zentralen „Master Reset“-Signal. Nach einem Selbsttest wird das Netzwerk gestartet und ein DHCP-Request abgesendet, um vom DHCP-Server der Management-Konsole eine IP-Adresse für den Zugriff auf das Netzwerk zu erhalten. Wurde die IP-Adresse erfolgreich übernommen, werden zwei weitere unabhängige Prozesse gestartet. Zum einen der sogenannte „Alive-Prozess“, der der Management-Konsole zyklisch den aktuellen Zustand der Adapter-Box meldet und zum anderen der „Treiber-Prozess“ passend für die angeschlossen Hardware. Im Initialen Teil der Treiberroutine wird die Kommunikation konfiguriert und ein optionaler Interrupt-Handler registriert. Je nach Gerät beendet sich dieser Prozess automatisch nach dem die Interrupt-Handler registriert worden sind, oder er wird erhalten, um ein Polling für das angeschlossene Gerät aus dem Treiber heraus zu ermöglichen. Nach dem Start der beiden Prozesse wird im Hauptpfad der Anwendung der Webserver für den darauf basierenden REST-Server gestartet. Ab jetzt startet die Hauptaufgabe der Schaltung: das Annehmen von Anfragen aus dem Netzwerk und das Ansteuern des entsprechenden Geräts oder das Weiterreichen der Nachrichten auf die anderen Netzwerke, falls die Adapter-Box als Gateway arbeitet.

Nach dem erfolgreichen Verarbeiten des Kommandos, das per Webservice empfangen worden ist, wird dem Empfänger auf dem gleichen Kanal die Antwort übermittelt. Resultiert aus dem empfangenen Kommando eine asynchrone REST Antwort, wird diese registriert, um bei Eintreffen des entsprechenden Signals versendet werden zu können.

Folgende Softwarekomponenten werden im Einzelnen benötigt:

- TCP/IP-Stack
- DHCP-Client
- HTTP-Server
- REST-Server
- HTTP-Client für asynchrone REST Anfragen
- Treiber für externe Peripherie
- Bibliotheken (Base64, MD5, CRC 32)
- Einfacher Multitasking Mechanismus

- Zugriffstreiber für die SD-Karte
- Selbsttestroutine für Bauteile auf der Adapter-Box
- Konsolenbasiertes Terminal für Debug-Ausgaben

4.8. Entwicklungsumgebung

Parallel zur Erstellung der Schaltung der Adapter-Box wurde mit dem Evaluationskit EVK100 der Firma Atmel die Software entwickelt.

Als Basis für die Anwendung diente das AVR32-SoftwareFramework-AT32UC3A-1.4.0, in welchem sich Beispielanwendungen für die freie Portierung des AVR32 GNU/GCC Compilers befinden. Für die Programmierung kann entweder der Adapter JTAG ICE mkII für die JTAG-Schnittstelle (Abbildung 44) oder das Softwaretool für den USB-Bootloader Flip verwendet werden. Falls der USB-Bootloader verwendet werden soll, muss das Programm auf Adresse 0x80002000 gelinkt werden.

Für Testausgaben und eine Debugkonsole steht eine serielle Schnittstelle zur Verfügung.

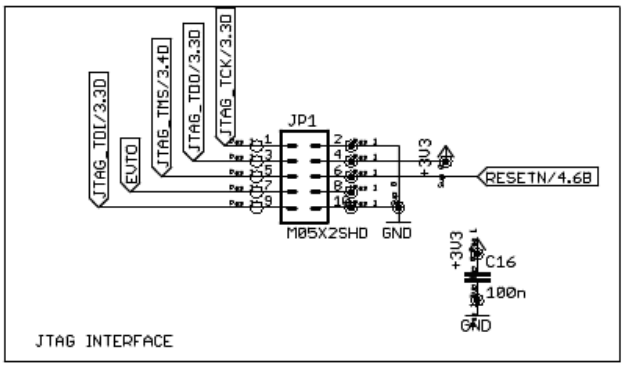


Abbildung 44: JTAG-Schnittstelle

Abbildung 45: JTAG ICE mkII Programmer und Debugger

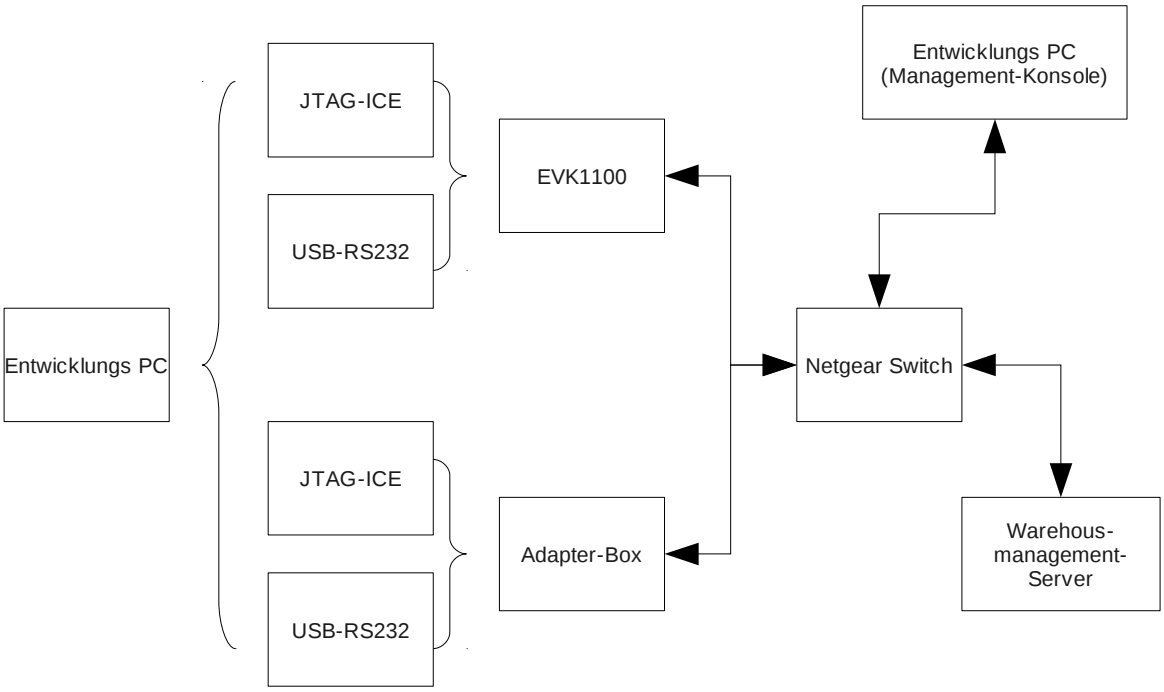


Abbildung 46: Entwicklungsumgebung

Abbildung 47: RS232-Wandler für Debugausgaben

Abbildung 48: EVK1100

4.9. Implementierung als FreeRTOS Anwendung

Als Basis für die Software wurde FreeRTOS eingesetzt. FreeRTOS ist ein portables Betriebssystem für Mikrocontroller. Im UC3 Software Framework findet sich eine angepasste Version mit bereits integriertem USB- und Netzwerkstack.

FreeRTOS bietet einen Multitasker an, welcher verschiedene Strategien fahren kann. Unterstützung für ein zentrales Interrupt-Handling. Mechanismen für die Synchronisation von Prozessen, Schnittstellen für die Integration von Hardware und Tools, die beim Entwickeln und Debuggen helfen.

FreeRTOS ist ideal für solch ein Projekt wie die Adapter-Box, da man sich direkt um die Implementierung des Ablaufes kümmern kann, ohne eine eigene Rahmenanwendung schreiben zu müssen.

Mit der FreeRTOS API können bequem neue Prozesse und Interrupt-Routinen, welche Daten über Queues abarbeiten, erstellt und verknüpft werden. Für weitere Informationen kann auf die Homepage www.freertos.org verwiesen werden.

Die Initialisierung der Anwendung ist in main.c Zusammengefasst.

```
(1)       /* Switch to external oscillator 0 */
(2)       pm_switch_to_osc0( pm, FOSC0, OSC0_STARTUP );
(3)
(4)       /* Setup PLL0 on OSC0, mul+1=16 ,
(5)          divisor by 1, lockcount=16, ie. 12Mhzx16/1 = 192MHz
output.
(6)        Extra div by 2 => 96MHz */
(7)       pm_pll_setup(pm,   /* volatile avr32_pm_t* pm */
(8)                    0,       /* pll */
(9)                    15,      /* mul */
(10)                   1,       /* div, Sel Osc0/PLL0 or Osc1/Pll1 */
(11)                   0,       /* osc */
(12)                   16);     /* lockcount */
(13)
(14)      pm_pll_set_option( pm, 0,   // pll0
(15)                             0,   // Choose the range 160-240MHz.
(16)                             1,   // div2
(17)                             0 ); // wbwdisable
(18)      /* Enable PLL0 */
(19)      pm_pll_enable(pm,0);
(20)
(21)      /* Wait for PLL0 locked */
(22)      pm_wait_for_pll0_locked(pm) ;
(23)
(24)      /* switch to clock */
```

```
(25)      pm_cksel( pm, 1, 1, 1, 0, 1, 0 );
(26)      flashc_set_wait_state( 1 );
(27)      pm_switch_to_clock( pm, AVR32_PM_MCCTRL_MCSEL_PLL0 );
(28)
(29)      /* Initialize the COM1 port. */
(30)      xComPort1Hndl = xUsartInit( serCOM1, 57600,
(31)                                    COM1SHELL_RXBUFFER_LEN,
COM1SHELL_TXBUFFER_LEN );
(32)      if(xComPort1Hndl == 0)
(33)      {
(34)        // xCom1ShellStatus = SYS_STATUS_DOWN;
(35)         vTaskDelete(NULL); // Kill this task.
(36)      }
(37)
(38)      sprintf(msg,"\r\nAdapter-Box Firmware %s\r\n",VERSION);
(39)      PrintMsg( xComPort1Hndl,msg);
(40)
(41)      vParDeviceInitialise();
(42)
(43)      vStartLEDFlashTasks( mainLED_TASK_PRIORITY );
(44)
(45)      vStartEthernetTaskLauncher( configMAX_PRIORITIES );
(46)
(47)      vTaskStartScheduler();
```

Im Wesentlichen sind das die Schritte:

- Einrichten Takt
- Initialisierung der Komponenten in der Adapter-Box
- Prozess für optische Ausgabe
- Logdatei Mechanismus starten
- Geräteansteuerungen aktiviere (für externe Geräte)
- Netzwerk-Stack starten,
- Alive-Prozess einrichten
- Starten alle initialisierten Prozesse

Auf der Debug-Konsole erscheint nach dem Einschalten folgende Ausgabe:

```
Adapter-Box Firmware 2.01
```

```
Done with DHCP: 192.168.0.104

ALIVE
ALIVE
ALIVE
```

Ist die Verbindung aktiv, geht die grüne LED an und leuchtet dauerhaft. Alle 10 Sekunden erfolgt die Ausgabe des Wortes „ALIVE“. Werden Nachrichten empfangen, erlischt die LED kurz für die Dauer der Ausgabe, damit die Aktivitäten optisch zu sehen sind.

4.10. LwIP Netzwerkstack

Der Netzwerkstack bildet die Basis der REST Implementierung. Für den eingesetzten Prozessor gibt es eine Portierung des bekannten und freien Open-Source Netzwerkstacks LwIP von Adam Dunkels [23] eine Portierung.

Die Struktur des Stacks ist nach den Schichten des OSI-Modells in Module aufgeteilt. Für den Betrieb wird ein Timer, ein Semaphor-Mechanismus, Funktionen zum Senden und Empfangen über eine Netzwerkschnittstelle und ein Speicherbereich, der intern über eine eigene Speicherverwaltung verwaltet wird, benötigt.

Die Hardware-Schnittstelle wird in ethernetif.c im Ordner lwip-port/netif implementiert. Der Treiber arbeitet direkt mit den FreeRTOS Mechanismen und Strukturen.

Weitere Informationen zu dem Stack können dem Handbuch [23] entnommen werden. Eigens erweitert wurde der Stack um das DHCP-Protokoll und einen Auto-IP Mechanismus. Ein Patch wurde manuell auf Basis der aktuellen Version von lwIP erstellt.

1. Änderungen an lwipopts.h:

   ändern:

   ```
   #define LWIP_DHCP 0
   ```

   in

   ```
   #define LWIP_DHCP 1

   #define MEMP_NUM_UDP_PCB 1
   ```

   in

   ```
   #define MEMP_NUM_UDP_PCB 2
   ```

   hinzufügen:

```
#define UDP_TTL                 255
#define LWIP_DHCP
#define LWIP_AUTOIP 1

#define LWIP_DHCP_AUTOIP_COOP 1
```

2. Anpassungen ethernet.c (nach prvlwIPInit();)

```
(1)        int mscnt =0;
(2)        while (MACB_if.ip_addr.addr==0) {
(3)                      sys_msleep(DHCP_FINE_TIMER_MSECS);
(4)                      dhcp_fine_tmr();
(5)                      mscnt += DHCP_FINE_TIMER_MSECS;
(6)                     if (mscnt >= DHCP_COARSE_TIMER_SECS*1000) {
(7)                             dhcp_coarse_tmr();
(8)                             mscnt = 0;
(9)                      }
(10)       }

(11)       PrintMsg( xComPort1Hndl,"\r\nDone with DHCP: ");
(12)              //print_dbg("\nAcquired IP =");
(13)              //
   print_dbg_ulong(0xff&(MACB_if.ip_addr.addr>>24));
(14)              //print_dbg(".");
(15)              //print_dbg_ulong(0xff&(MACB_if.ip_addr.addr>>16)
   );
(16)              //print_dbg(".");
(17)              //print_dbg_ulong(0xff&(MACB_if.ip_addr.addr>>8))
   ;
(18)              //print_dbg(".");
(19)       //print_dbg_ulong(0xff&(MACB_if.ip_addr.addr));
(20)       portCHAR cPageHits[ 20 ];
(21)
(22)         sprintf( cPageHits, "%d.%d.%d.%d\r\n",
   (int)0xff&(MACB_if.ip_addr.addr>>24),
(23)                  (int)0xff&(MACB_if.ip_addr.addr>>16),
   (int)0xff&(MACB_if.ip_addr.addr>>8),
   (int)0xff&(MACB_if.ip_addr.addr));
(24)         PrintMsg( xComPort1Hndl,cPageHits);
```

3. Dateien hinzufügen

   autoip.c, dhcp.c

4. prvEthernetConfigureInterface anpasen (feste IP entfernen)

```
(1)        MacAddress[0] = ETHERNET_CONF_ETHADDR0;
(2)        MacAddress[1] = ETHERNET_CONF_ETHADDR1;
(3)        MacAddress[2] = ETHERNET_CONF_ETHADDR2;
(4)        MacAddress[3] = ETHERNET_CONF_ETHADDR3;
(5)        MacAddress[4] = ETHERNET_CONF_ETHADDR4;
```

```
(6)         MacAddress[5] = ETHERNET_CONF_ETHADDR5;
(7)
(8)         vMACBSetMACAddress( MacAddress );
(9)
(10)        IP4_ADDR( &xIpAddr,0,0,0,0 );
(11)        IP4_ADDR( &xNetMask,0,0,0,0 );
(12)        IP4_ADDR( &xGateway,0,0,0,0 );
(13)
(14)        netif_add( &MACB_if, &xIpAddr, &xNetMask, &xGateway,
   NULL, ethernetif_init, tcpip_input );
(15)        netif_set_default( &MACB_if );
(16)
(17)        dhcp_start(&MACB_if);
```

Direkt nach dem Einschalten versucht die Adapter-Box eine IP-Adresse per DHCP zu bekommen. Schlägt dies Fehl, da kein DHCP-Server vorhanden ist, wird mittels Auto-IP Mechanismus versucht, eine gültige IP-Adresse zu erlangen (nach RFC 3927).

Die Adapter-Box muss sich zyklisch (alle 10 Sekunden) im aktiven Zustand an der Management-Konsole melden. Hierfür muss eine TCP-Verbindung zu dem Port der Management-Konsole mittels HTTP aufgebaut werden:

```
(1)     portTASK_FUNCTION( vBasicAliveServer, pvParameters ){
(2)     struct netconn *pxHTTPListener, *pxNewConnection;
(3)     portTickType xFlashRate, xLastFlashTime;
(4)
(5)         vParTestSetLED(webCONN_LED, pdTRUE);
(6)
(7)          xLastFlashTime = xTaskGetTickCount();
(8)
(9)         /* Loop forever */
(10)        for( ;; )
(11)        {
(12)             vTaskDelayUntil(&xLastFlashTime,10000);
(13)
(14)            //tcp_connect(pcb, &target, 8080, &connected_callback);
(15)             //PrintMsg( xComPort1Hndl,"ALIVE\r\n");
(16)             vSendAlive();
(17)
(18)        } /* end infinite loop */
(19)    }
(20)
```

Der Prozess vBasicAliveServer ruft alle 10 Sekunden die Funktion vSendAlive auf, welche eine TCP-Verbindung zum Server aufbaut und dort einen REST Aufruf startet. Nach dem Aufruf legt sich der Prozess für 10 Sekunden schlafen um die Rechenzeit für weitere Prozesse freizugeben.

```
(1)     void vSendAlive( )
(2)     {
(3)         struct ip_addr  target;
(4)
(5)
(6)         IP4_ADDR(&target, 192,168,0,107);
(7)          struct tcp_pcb *pcb = tcp_new();
(8)          tcp_connect(pcb, &target, 8080, &connected_callback);
(9)
(10)         PrintMsg( xComPort1Hndl,"ALIVE\r\n");
(11)
(12)    }
```

Mittels tcp_write wird der Inhalt der Nachricht versendet, nach dem tcp_connect die Verbindung aufgebaut hat.

```
(1)     /* Connection etablished, lets try to get a http page */
(2)     err_t connected_callback(void *arg, struct tcp_pcb *pcb,
   err_t err) {
(3)         //tcp_recv(pcb, &recv_callback);
(4)         char msg[] = "GET /alive/waage01 HTTP/1.1\r\nHost:
   192.168.0.107\r\n\r\n";
(5)         if (tcp_write(pcb, msg, strlen(msg), 1) != ERR_OK) {
(6)             PrintMsg(xComPort1Hndl,"Could not write");
(7)         }
(8)         return ERR_OK;
(9)     }
```

Dieser Auszug zeigt die Passage im Quelltext. In der endgültigen Version wird das Label aus einem internen Speicher gelesen und die Adresse des Server dynamisch ermittelt.

Die folgende Funktion CommandExecute entscheidet, ob es sich um ein intern oder extern angebundenes Label handelt, für das die Nachricht bestimmt ist. Bei einem externen wird diese mittels CommandGateway weiterverarbeitet. Interne Kommandos werden direkt verarbeitet bzw. ausgeführt und die Antwort an den Sender zurückgesendet. Handelt es sich um eine asynchrone Nachricht, wird dies intern vermerkt und bei vorhanden sein der Antwort übermittelt.

```
(1)     void CommandExecute(struct netconn *pxNetCon, portCHAR
   *pcRxString)
(2)     {
```

```
(3)         portTickType xFlashRate, xLastFlashTime;
(4)
(5)         if(ExternLabel(pcRxString))
(6)         {
(7)              CommandGateway( pxNetCon, pcRxString);
(8)              return;
(9)         }
(10)
(11)        if(AsyncMessage(pcRxString))
(12)        {
(13)             RegisterAsyncMessage( pxNetCon, pcRxString);
(14)        }
(15)
(16)
(17)        /* Write out the HTTP OK header. */
(18)        netconn_write( pxNetCon, webHTTP_OK, (u16_t)
                      strlen( webHTTP_OK ), NETCONN_COPY );
(19)
(20)        // LED
(21)        if(strstr(pcRxString,"ledblink")!=NULL)
(22)        {
(23)    netconn_write( pxNetCon,"STARTrunEND",10,NETCONN_COPY );
(24)
(25)             xLastFlashTime = xTaskGetTickCount();
(26)            int i;
(27)            for(i=0;i<3;i++)
(28)            {
(29)             vTaskDelayUntil(&xLastFlashTime,500);
(30)             vParTestSetLED(webCONN_LED, pdFALSE);
(31)             vTaskDelayUntil(&xLastFlashTime,500);
(32)             vParTestSetLED(webCONN_LED, pdTRUE);
(33)            }
(34)        }
(35)        // GEWICHT
(36)        else if(strstr(pcRxString,"gewicht")!=NULL)
(37)        {
(38)
       netconn_write( pxNetCon,"START5.000END",13,NETCONN_COPY );
(39)        }
(40)        else if(strstr(pcRxString,"label")!=NULL)
(41)        {
(42)

   netconn_write( pxNetCon,"STARTwaage01END",15,NETCONN_COPY );
(43)        }
(44)
(45)        PrintMsg( xComPort1Hndl,pcRxString);
(46)
(47)    }
```

4.11. Emulator für die Entwicklung

Im Ordner „emulator“ auf der mitgelieferten CD befindet sich ein in Python geschriebener Adapter-Box Emulator. Für die Entwicklung der Management-Konsole und Adapter-Box war diese Software sehr nützlich. Der Emulator kann in Zukunft weiter für die Entwicklung als Basis für neue Gerätetreiber eingesetzt werden. Mit minimalem Aufwand kann auf diese Weise ein Gerät für die Software emuliert werden, um alle Algorithmen in der Anwendung entwickeln und testen zu können.

Entsprechend dem Ablauf der Anwendung (Abbildung 43) existieren die folgenden Funktionen:

def Selftest(self)

Selbsttest für die auf der Schaltung befindlichen Bauteile

def DHCPRequest(self)

Absenden eines DHCP-Requests an den DHCP-Server der Management-Konsole

def AliveProcess(self)

Eigener Thread für Alive-Prozess nach Registrierung

def DeviceInit(self)

Emulation des Aufrufs für die Initialisierung es angeschlossen Geräts

def RegisterOnManagementKonsole(self):)

Anmeldeprozedur der Adapter-Box an Management-Konsole

def StartREST()

Start des REST-Servers für Kommandoverarbeitung

def ParseKommando()

Parsen und Überprüfen der Prüfsumme des eingehenden Kommandos

def GatewayKommando()

Weiterleitung eines Kommandos an eine interne Schnittstelle

def DeviceKommando()

Verarbeitung eines Gerätekommandos

4.12. Hintergrunddienst für die PC-Integration

Wie bereits erwähnt sollen auch Standard Computer auf die gleiche Art und Weise in das Netzwerk der Management-Konsole integriert werden, wie die Adapter-Box. Hierfür wurde der Emulator erweitert und um die Kommandos „shutdown“ mit einem passenden Skript ergänzt. Die Version wurde unter Berücksichtigung auf gute Portierbarkeit auf einem Ubuntu 9.04 System entwickelt.

Aufgaben des Hintergrunddienstes:

- Anmeldung an Management-Konsole
- Erzeugung „Alive Signal“
- Kommando Shutdown für Computer
- Eigenes Label verwalten

Unterschied zum Verhalten der Adapter-Box:

- Erst prüfen der Verbindung zur Management-Konsole
- Wenn keine Verbindung, dann DHCP-Request senden

Installation des Hintergrunddienstes:

- Kopieren der Datei nach /usr/sbin
- Einhängen in Startprozess des Computer z.B. /etc/rcS.d/

Konfiguration:

- Label wird aus Computername erzeugt
- Wake on LAN einrichten (siehe nächster Abschnitt)

Der Grundrahmen der Anwendung ist in Python überschaubar:

```
(1)     import time
(2)     import threading
(3)     import web
(4)     import httplib
(5)     import socket
(6)     import os
(7)
(8)     urls = (
(9)         '/info(/)?', 'Info',
(10)        '/power/(.*)', 'Power'
(11)    )
(12)    app = web.application(urls, globals())
(13)
(14)
(15)
(16)    class Info:
(17)        def GET(self, name):
(18)            if not name:
(19)                name = socket.gethostname()
(20)            return 'InfoHello, ' + name + '!'
(21)
(22)
(23)    class Power:
(24)        def GET(self, name):
(25)            if name == 'off':
(26)              print "Shutdown " + name
(27)              os.system("/sbin/halt")
(28)              return 'New State: ' + name + '!'
(29)            else:
(30)              return 'New State: unknown'
(31)
(32)
(33)    class Timer(threading.Thread):
(34)        def __init__(self, seconds, action):
(35)            self.runTime = seconds
(36)            self.action = action
(37)            threading.Thread.__init__(self)
(38)        def run(self):
(39)            time.sleep(self.runTime)
(40)            self.action()
(41)
(42)
(43)    def myTick():
(44)        # do job here
(45)        conn = httplib.HTTPConnection("192.168.0.107",8080)
(46)        conn.request('GET', '/alive/' + socket.gethostname())
(47)
(48)        print "."
(49)        c = Timer(10,myTick)
(50)        c.start()
(51)
```

```
(52)    if __name__ == "__main__":
(53)        c = Timer(10,myTick)
(54)        c.start()
(55)        app.run()
```

In main wird ein Timer initialisiert, welcher alle 10 Sekunden aufgerufen wird. In dem Aufruf des Timers wird ein REST-Aufruf zur Management-Konsole abgesendet, in dem mitgeteilt wird, dass das Gerät aktiv im Netzwerk hängt. Als Label wird der im Computer eingetragene Computername verwendet. Aktuell ist im Hintergrunddienst nur die Fähigkeit eingebaut, den Computer mittels REST-Aufruf herunterzufahren. Gestartet kann er über die Management-Konsole werden, welche Wake on LAN Nachrichten versenden kann.

Einrichten des Hintergrunddienstes:

1. sudo apt-get install python-webpy
2. Kopieren der Datei adapter-box-emulator.py
3. sudo chmod +s /sbin/halt
4. sudo chmod +x adapter-box-emulator.py
5. Startskript nach /etc/init.d/adapter-box.sh

```
#!/bin/sh

WDIR=/home/eproo

case "$1" in
start)
echo -n "Starting Mini-Browser"
cd $WDIR; python adapter-box-emulator.py &
echo "."
;;
esac

exit 0
```

6. Symlink ln -s /etc/init.d/adapter-box.sh /etc/rc2.d/S89adapter-box

4.13. Wake on LAN an Station

Um Wake on LAN auf einem herkömmlichen Linux System nutzen zu können, muss dies aktiviert werden.

Mit Hilfe des Programms „ethtool“ kann auf einen GNU/Linux System überprüft werden, ob WoL von der Netzwerkkarte unterstützt wird.

```
stations-pc ~ # ethtool eth0
Settings for eth0:
     Supported ports: [ TP MII ]
     Supported link modes:   10baseT/Half 10baseT/Full
                             100baseT/Half 100baseT/Full
     Supports auto-negotiation: Yes
     Advertised link modes:  10baseT/Half 10baseT/Full
                             100baseT/Half 100baseT/Full
     Advertised auto-negotiation: Yes
     Speed: 100Mb/s
     Duplex: Full
     Port: MII
     PHYAD: 8
     Transceiver: internal
     Auto-negotiation: on
     Supports Wake-on: pumbg
     Wake-on: d
     Current message level: 0x00000001 (1)
     Link detected: yes
```

Die vorliegende Karte unterstützt die Optionen **p**, **u**, **m**, **b** und **g**.

```
p  Wake on phy activity
u  Wake on unicast messages
m  Wake on multicast messages
b  Wake on broadcast messages
a  Wake on ARP
g  Wake on MagicPacket(tm)
s  Enable SecureOn(tm) password for MagicPacket(tm)
d  Disable (wake on nothing).  Clear all previous options.
```

Mit dem Parameter wol muss nur noch die passende Option gesetzt werden. Um mit dem MagicPacket arbeiten zu können, bedeutet dies, die Option g muss gesetzt sein.

```
stations-pc ~ # ethtool -s eth0 wol g
```

Dieser Parameter muss für zukünftige Sitzungen immer beim Start gesetzt werden. Am einfachsten wird dies in einem eigenem Skript aktiviert, welches beim Starten automatisch aufgerufen wird.

```
(1)  #!/sbin/runscript
(2)
(3)  depend() {
(4)      need net
(5)  }
(6)
(7)
(8)  start() {
(9)      ebegin "Starting WakeUpOnLan..."
(10)     /usr/sbin/ethtool -s eth0 wol g
(11)     eend $? "Failed to start ethtool"
(12) }
(13)
(14) stop() {
(15)     ebegin "activate Wake Up On Lan"
(16)     /usr/sbin/ethtool -s eth0 wol g
(17)     eend $? "Failed to start ethtool"
(18) }
```

Jetzt kann der Computer, wenn das Netzwerk steht, bequem von extern gestartet werden. In der Management-Konsole sind Bibliotheken integriert um „Magic Pakete“ zu versenden (Abschnitt 5.4. Seite 93).

5. Softwarearchitektur der Management-Konsole

Nach dem nun die Anbindung der Geräte mit der Adapter-Box per Netzwerk möglich ist, fehlt noch die Beschreibung der Implementierung der Managament-Konsole als zentrale Komponente für die Registrierung und Verwaltung aller Geräte und Computer.

Der zentrale Ansatz bietet den Vorteil, die Managementaufgaben gebündelt und dadurch kontrolliert von einer Instanz ablaufen bzw. Informationen zentral sammeln zu können. Dieser Gedanke widerspricht jedoch dem REST Konzept, kann aber mittels Hochverfügbarkeitsalgorithmen schnell als eine neue Instanz auf der IP-Adresse der Managenent-Konsole gestartet werden, falls diese nicht mehr reagiert. In einer weiteren Lösung könnte ein per Round-Robin implementiertes oder ähnliches Verfahren Anfragen auf verschiedene Management-Konsolen verteilen. Die Management-Konsolen könnten über ein P2P Netzwerk in Kontakt stehen.

Über eine REST Schnittstelle sind ebenfalls die Funktionen der Management-Konsole für alle anderen Teilnehmer erreichbar. Das Übersichtsbild (siehe Abbildung 49) stellt nochmals die Position der Konsole im Netzwerk dar. Vergleichbar mit einem Proxy bzw. Gateway werden die Kommunikationsstrecken verknüpft. bzw. von anderen Netzwerken aus erreichbar gemacht.

Die Management-Konsole hat eine größere Liste an Anforderungen (siehe Absatz 5.1). Zusätzlich soll die Software auf einem eigenem Embedded System lauffähig sein. Dadurch kann die Konsole einfach Installiert und gewartet werden (Austausch des Geräts) und zudem wird der Stromverbrauch gesenkt, da hierfür kein eigener Computer verwendet werden muss.

Zusammengefasst ist die Management-Konsole im Wesentlichen eine Software, die als klassischer Netzwerkdienst implementiert ist, aber von den Ressourcen so Gebrauch machen soll, dass sie ebenso auf einem Embedded GNU/Linux wie auch als weiterer Dienst auf einem PC-Server bzw. einem Firmenserver ablaufen kann.

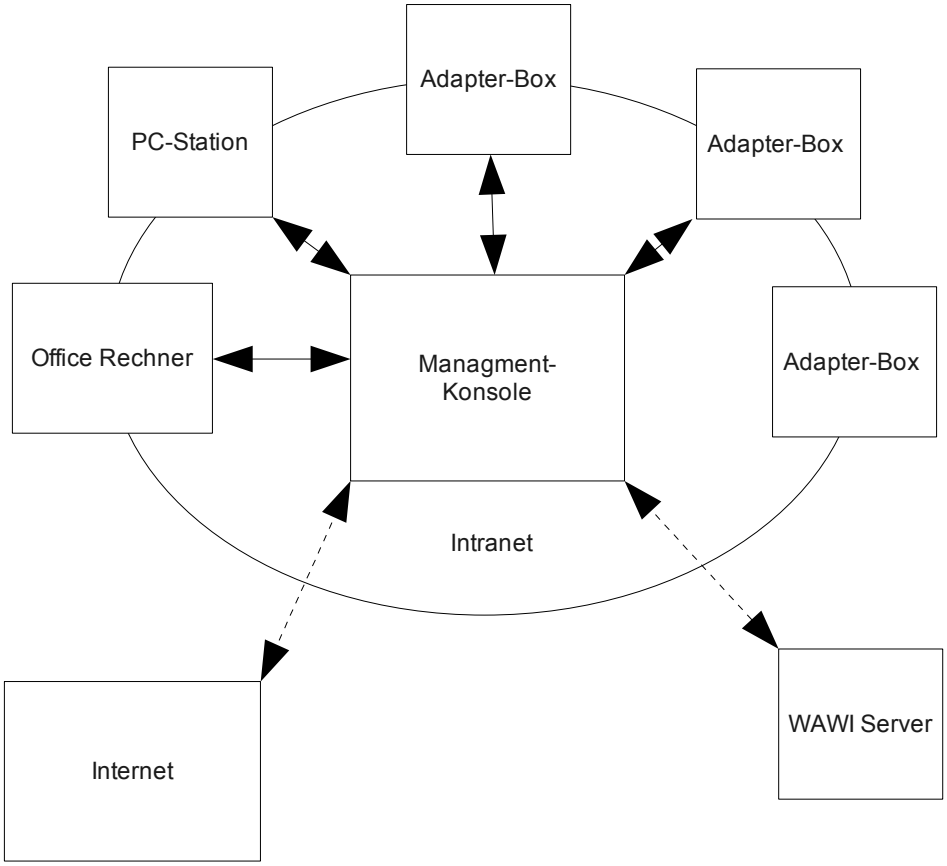


Abbildung 49: Übersichtsbild Netzwerkstruktur

5.1. Anforderungen

Die zentrale Verwaltung der Geräte ergibt verschiedenste Aufgaben und Protokolle, welche implementiert sein müssen. Es folgt eine stichpunktartige Auflistung der Anforderungen, auf welche anschließend im einzelnen eingegangen wird:

- Auto Power Management (Zeit- und Ereignisgesteurtes Power-Management)
- DHCP Server (für die Vergabe der IP-Adressen des Verwaltungsnetzwerkes)
- Lokaler DNS Server (für die Auflösung der Labels der Teilnehmer zu den IP-Adressen)
- Rückgabe der IP für passendes Label (mittels REST Aufruf)

- Entgegennahme der Befehle von der Anwendung (Proxy)
- Knoten für Anfragen aus anderen Netzwerken (Gateway)
- Verwaltung der Teilnehmerliste
- Einfache Konsole bzw. GUI für Benutzer per Telnet oder HTTP
- API für externe GUI (mittels REST)
- REST Server
- Alive Signale der Teilnehmer sammeln und auswerten
- Integration der Anwendung in ein Embedded System ähnlich wie Router oder Firewalls

Auto Power Management (Zeit- und Ereignisgesteuertes Power-Management)

Computer und Geräte müssen über die Konsole ein- und ausgeschaltet werden können. Zum einen soll dies intern über eine „Zeitschaltuhr“ und zum anderen per REST Aufruf möglich sein. Die zentrale Zeit gesteuerte Lösung ist vor allem für die Anwendungsfälle Arbeitsbeginn oder Arbeitsende interessant. Alle Hauptstationen können morgens zentral gestartet und abends nach Verlassen der jeweiligen Station (soweit kein Benutzer angemeldet ist) automatisch wieder heruntergefahren werden. Für das Ein- und Ausschalten der Komponenten gibt es verschiedene Möglichkeiten:

- Einschalten: Mittels Wake On LAN oder Adapter-Box
- Ausschalten: Computer per Adapter-Box Emulation und externe Geräte mittels Adapter-Box

DHCP Server (für die Vergabe der IP-Adressen des Verwaltungsnetzwerkes)

Alle Teilnehmer kommunizieren über ein eigenes Netzwerk. Als Standardeinstellung wird das Netzwerk 10.0.0.0 verwendet. Bevor ein Teilnehmer Mitglied wird, muss er zuvor eine gültige IP-Adresse anfordern. Nachdem die IP-Verbindung steht, erfolgt jede weitere Kommunikation zur Management Konsole, welche in der Regel auf der Adresse 10.0.0.1 hört.

Lokaler DNS Server (für die Auflösung der Labels der Teilnehmer zu den IP-Adressen) bzw. Rückgabe der IP-Adresse für das entsprechende Label

Um aus der Anwendung heraus mit den Teilnehmern des Netzwerks kommunizieren zu können wird die entsprechende URL benötigt. Entweder wird mittels lokalem DNS Server das Label des Teilnehmers als Hostname aufgelöst oder es ist mittels REST Aufruf auf der Konsole möglich, die IP-Adresse des gewünschten Labels zu erfragen, um dann mit der IP-Adresse direkt den REST Aufruf an den Teilnehmer zu übertragen.

Entgegennahme der Befehle von Anwendung (Proxy)

Anwendungen können mit den Teilnehmern des Netzwerks immer direkten Kontakt aufnehmen. Jeder Teilnehmer kann abhängig von seinen Ressourcen beliebig viele Kommunikationen abarbeiten. Soll jedoch nicht direkt mit dem Teilnehmer kommuniziert werden, kann ebenso über die Konsole die Anfrage abgesendet werden. Für solche Anfragen bietet die Konsole einen asynchronen REST Mechanismus an, um die Anfrage transparent für den Teilnehmer und die Anwendung abbilden zu können.

Knoten für Anfragen aus anderen Netzwerken (Gateway)

Befinden sich Teilnehmer oder PC-Stationen in anderen Netzwerken (z.B. dem Firmennetzwerk oder dem Internet), so kann über einen REST Gateway dennoch kommuniziert werden.

Ebenfalls könnten hier verschiedene Schnittstellen (falls in der Hardware der Konsole vorhanden) verarbeitet werden.

Verwaltung der Teilnehmerliste

Über interne Datenstrukturen müssen Netzwerkteilnehmer für Anwendungen jederzeit abfragbar sein. Werden Teilnehmer entfernt oder hinzugefügt, bedeutet dies eine Restrukturierung der Listen. Die Liste muss auch nach einem Stromausfall vorhanden sein, um das Netzwerk mittels Wake on LAN wieder starten zu können.

Einfache Konsole bzw. GUI für Benutzer per Telnet oder HTTP

Die Darstellung der internen Zustände, Aktivitäten und Funktionen der Managementkonsole ist Aufgabe der Anwendung für die das Netzwerk betrieben wird. Um jedoch für die Administration schnellen und einfachen Einblick zu ermöglichen, soll es eine einfache Oberfläche oder Schnittstelle zur Anwendung geben.

API für externe GUI (mittels REST)

Um in der Anwendung eine grafische Ausgabe zu ermöglichen, gibt es eine API, welche hierfür alle notwendigen Daten und Algorithmen anbietet.

REST Server

Zentrale Komponente des Netzwerks ist die Kommunikation mittels REST über HTTP. Um die Anfragen entgegen nehmen zu können dient der REST-Server.

Alive Signale der Teilnehmer sammeln und auswerten

Periodisch müssen die Teilnehmer der Konsole den eigenen Status übermitteln. Die Konsole benötigt hierfür Ressourcen schonende Einheiten.

Integration der Anwendung in ein Embedded System ähnlich wie Router oder Firewalls

Die Konsole soll flexibel auf PC-Servern oder Embedded Systemen installiert werden können. Als Basis wird immer ein GNU/Linux System mit Python Support benötigt.

5.2. Implementierung in Python

Die Management-Konsole ist in Python geschrieben. Python ist eine beliebte und weit verbreitete Skriptsprache. In nur wenigen Zeilen lassen sich aufwändige Aufgaben lösen:

- Große Vielfalt an Bibliotheken (REST, Cron, WOL, web.py)
- Portierbar auf Embedded Systeme
- Einfach zu installieren in GNU/Linux Systeme (Paketmanager)
- Open-Source

Als Basis dient das Framework web.py. (http://webpy.org/). Das Framework hilft mit minimalem Aufwand Netzwerkanwendungen - speziell Webanwendungen - schnell und einfach zu implementieren. Der Webserver von web.py bietet sich zudem Ideal für die Abbildung eines REST-Servers an.

```
(1)     import time
(2)     import threading
(3)     import web
```

Die URLs geben die Ressourcen an, welche der Server zur Verfügung stellt.

```
(4)
(5)     urls = (
(6)         '/devices(/)?', 'Devices',
(7)         '/devices/(.*)', 'Device',
(8)         '/alive/(.*)', 'Alive'
(9)     )
(10)    app = web.application(urls, globals())
(11)
(12)    // Hier kommen die REST Ressourcen hin
(13)
(14)
(15)    class Timer(threading.Thread):
```

```
(16)        def __init__(self, seconds, action):
(17)            self.runTime = seconds
(18)            self.action = action
(19)            threading.Thread.__init__(self)
(20)        def run(self):
(21)            time.sleep(self.runTime)
(22)            self.action()
(23)
(24)
(25)    def myTick():
(26)        # do job here
(27)        print "."
(28)        c = Timer(1,myTick)
(29)        c.start()
(30)
(31)    if __name__ == "__main__":
(32)        #c = Timer(1,myTick)
(33)        #c.start()
(34)        app.run()
(35)
```

Gestartet wird der Server über die Kommandozeile:

python managemen-konsole.py

Wird als Parameter ein optionaler Port angeben, startet der Server auf diesem und nicht vom Standard 8080.

Exemplarische Ausgabe der Management-Konsole. Der Computer „bene-laptop“ und das Gerät „waage01“ befinden sich aktiv im Netzwerk und können verwendet werden.

```
192.168.0.107:42329 - - [07/Jun/2010 07:47:37] "HTTP/1.1 GET /alive/bene-laptop" - 200 OK
192.168.0.107:42330 - - [07/Jun/2010 07:47:38] "HTTP/1.1 GET /devices" - 200 OK
192.168.0.107:42332 - - [07/Jun/2010 07:47:38] "HTTP/1.1 GET /devices" - 200 OK
192.168.0.112:4560 - - [07/Jun/2010 07:47:43] "HTTP/1.1 GET /alive/waage01" - 200 OK
192.168.0.107:42334 - - [07/Jun/2010 07:47:47] "HTTP/1.1 GET /alive/bene-laptop" - 200 OK
192.168.0.112:4561 - - [07/Jun/2010 07:47:53] "HTTP/1.1 GET /alive/waage01" - 200 OK
192.168.0.107:42335 - - [07/Jun/2010 07:47:57] "HTTP/1.1 GET /alive/bene-laptop" - 200 OK
```

5.3. DHCP Server

Für die automatische Vergabe der IP-Adressen für die Teilnehmer dient ein DHCP-Server. Jeder Client muss so ebenfalls einen DHCP-Client haben. Als DHCP-Server kann eine Standardsoftware wie z.B. udhcpd eingesetzt werden.

Über die Paketverwaltung kann die Software installiert werden:

sudo apt-get install udhcpd

Welche IP-Adressen vergeben werden, auf welchem Netzwerkinterface der DHCP-Server hört und viele weitere Optionen können in der Konfigurationsdatei angeben werden.

```
(1)     #/etc/udhcpd.conf
(2)     start   192.168.0.1
(3)     end     192.168.0.254
(4)     interface eth0
(5)
(6)     lease_file /usr/lib/udhcpd/udhcpd.leases
(7)     remaining yes
(8)     option subnet 255.255.255.0
(9)     option domain home
(10)    option lease 3600
```

Alternativ kann die Funktion des DHCP Servers direkt als Python Modul in die Management-Konsole integriert werden. Hierfür bietet sich das Projekt ameon (http://anemon.org/pmwiki/) an.

5.4. Wake on LAN

Die Teilnehmer des Netzwerkes können per Wake on LAN - wenn vorhanden - eingeschaltet werden. Zentral kann dies über die Management-Konsole geschehen. Eine Implementierung wurde direkt in Python integriert.

Im Wesentlichen wird das sogenannte „MagicPaket“[24] erzeugt, welches per Ethernet in das Netzwerk übertragen werden kann und von den Teilnehmern ausgewertet werden kann.

Ein MagicPaket besteht aus sechs mal 0xFF und anschließend 16 mal der MAC Adresse, von dem Teilnehmer welcher gestartet werden soll. Versendet wird das Paket als Broadcast, um so von allen Teilnehmern empfangen werden zu können.

```
(1)     def wake_on_lan(macaddress):
(2)         """ Switches on remote computers using WOL. """
(3)
```

```
(4)         # Check macaddress format and try to compensate.
(5)         if len(macaddress) == 12:
(6)             pass
(7)         elif len(macaddress) == 12 + 5:
(8)             sep = macaddress[2]
(9)             macaddress = macaddress.replace(sep, '')
(10)        else:
(11)            raise ValueError('Incorrect MAC address format')
(12)
(13)        # Pad the synchronization stream.
(14)        data = ''.join(['FFFFFFFFFFFF', macaddress * 20])
(15)        send_data = ''
(16)
(17)        # Split up the hex values and pack.
(18)        for i in range(0, len(data), 2):
(19)            send_data = ''.join([send_data,
(20)                    struct.pack('B', int(data[i: i + 2], 16))])
(21)
(22)        # Broadcast it to the LAN.
(23)        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
(24)        sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST,
(25)        sock.sendto(send_data, ('<broadcast>', 7))
```

Über die REST-Ressource /devices/{label}/on bzw. /devices/{label}/off kann der Teilnehmer über die Management-Konsole ein- und ausgeschaltet werden.

```
(1)     class Device:
(2)         def GET(self, name):
(3)             name_array = name.split("/")
(4)             if name_array[1]=="on":
(5)               print "Einschalten " + name
(6)               if name_array[0] =="theke":
(7)                 wake_on_lan('00:15:58:7a:7a:7f')
(8)             if name_array[1]=="off":
(9)               print "Ausschalten " + name
(10)              for key, value in devices.items():
(11)                if value.label==name_array[0]:
(12)                  print "request"
(13)                  conn = httplib.HTTPConnection(value.ip,8080)
(14)                  conn.request('GET', '/power/off')
(15)            return 'Device ' + name + '!'
```

Alternativ kann über das konsolenbasierte Programm wakeonlan ebenfalls ein Magic-Paket erzeugt und versendet werden.

5.5. Geräteverwaltung

Zu den wesentlichen Aufgaben der Management-Konsole gehört das Führen der Teilnehmer in einer internen Datenstruktur für weitere REST-Ressourcen. Jeder Teilnehmer meldet sich regelmäßig mittels „Alive-Signal“ an der Management-Konsole. Meldet sich das Gerät zum ersten Mal, wird eine neue Instanz der Klasse DeviceInfo erstellt und in dem Dictionary devices abgelegt. Ergeben sich Änderungen am Gerät, kann dies ebenfalls durch das Alive-Signal festgestellt und entsprechend geändert werden (Label oder IP-Adresse). Zudem wird bei jedem „Alive-Signal“ ein Timestamp aktualisiert. Dadurch kann die Management-Konsole feststellen, wenn ein Gerät ausfällt bzw. keine Kommunikation stattfinden kann, da kein gültiges Signal vorliegt.

```
(1)     class DeviceInfo():
(2)         def __init__(self, label):
(3)             self.label = label
(4)             self.timestamp="none"
(5)             self.ip ="none"
(6)
(7)
(8)     devices = {}
(9)
(10)    class Devices:
(11)        def GET(self,name):
(12)          collect = "START"
(13)          for key, value in devices.items():
(14)            collect = collect + key + ";" + value.timestamp +
   ";" + value.ip + "!"
(15)          collect = collect + "END"
(16)
(17)
(18)          return collect
(19)
(20)    class Alive:
(21)        def GET(self, name):
(22)            if devices.has_key(name):
(23)              devices[name].timestamp = str(time.time())
(24)              devices[name].ip = web.ctx.ip
(25)            else:
(26)              print "Create new device "+ name
(27)              devices[name] = DeviceInfo(name)
(28)              devices[name].timestamp = str(time.time())
(29)              devices[name].ip = web.ctx.ip
(30)
(31)            if not name:
(32)                name = 'World'
(33)            return 'Alive ' + name + '!'
```

5.6. DNS Server

Die Anwendung, welche das Netzwerk nutzt, hat verschiedene Möglichkeiten, die Ressourcen der Teilnehmer anzusprechen. Entweder wird die HTTP-Anfrage direkt an die IP-Adresse des Teilnehmers, an die Management-Konsole oder eben über eine URL, die durch die Management-Konsole aufgelöst werden muss, abgesendet. Die Implementierung des DNS-Server ist in Python ebenfalls mit ein paar wenigen Zeilen rudimentär erledigt. Der folgende Quelltext zeigt ein Beispiel für die Implementierung direkt auf dem Modul socket.

```
(1)     import socket
(2)
(3)     class DNSQuery:
(4)       def __init__(self, data):
(5)         self.data=data
(6)         self.dominio=''
(7)
(8)         tipo = (ord(data[2]) >> 3) & 15   # Opcode bits
(9)         if tipo == 0:                     # Standard query
(10)          ini=12
(11)          lon=ord(data[ini])
(12)          while lon != 0:
(13)            self.dominio+=data[ini+1:ini+lon+1]+'.'
(14)            ini+=lon+1
(15)            lon=ord(data[ini])
(16)
(17)      def respuest(self, ip):
(18)        packet=''
(19)        if self.dominio:
(20)          packet+=self.data[:2] + "\x81\x80"
(21)          packet+=self.data[4:6] + self.data[4:6] +
   '\x00\x00\x00\x00'   # Questions and Answers Counts
(22)          packet+=self.data[12:]
   # Original Domain Name Question
(23)          packet+='\xc0\x0c'
   # Pointer to domain name
(24)          packet+='\x00\x01\x00\x01\x00\x00\x00\x3c\x00\x04'
   # Response type, ttl and resource data length -> 4 bytes
(25)          packet+=str.join('',map(lambda x: chr(int(x)),
   ip.split('.'))) # 4bytes of IP
(26)        return packet
(27)
(28)    if __name__ == '__main__':
(29)      ip='192.168.0.104'
(30)      print 'pyminifakeDNS:: dom.query. 60 IN A %s' % ip
(31)
(32)      udps = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
(33)      udps.bind(('',53))
(34)
(35)      try:
```

```
(36)        while 1:
(37)          data, addr = udps.recvfrom(1024)
(38)          p=DNSQuery(data)
(39)          udps.sendto(p.respuest(ip), addr)
(40)          print 'Respuest: %s -> %s' % (p.dominio, ip)
(41)      except KeyboardInterrupt:
(42)        print 'Finalizando'
(43)        udps.close()
```

Alternativ kann ein fertiger DNS Server verwendet werden, der bei Änderungen dynamisch umkonfiguriert wird.

5.7. Eingebettete Lösung integriert in das Netzwerk

Wie bereits erwähnt, kann die Management-Konsole auf einem eigenen Embedded System installiert sein. Der Vorteil ist, dass so alle Komponenten des Netzwerks einfach austauschbar und somit keine Wartung durch die Administration benötigt wird.

Ein populäres Entwicklungsboard, welches sich ideal als Plattform für die Management-Konsole anbietet, ist das NGW100 von Atmel. Es bietet einen modernen 32-Bit Prozessor, 32 MB Arbeitsspeicher,FLASH, DATAFLASH, einen externen SD-Kartenhalter und zwei Netzwerkschnittstellen.

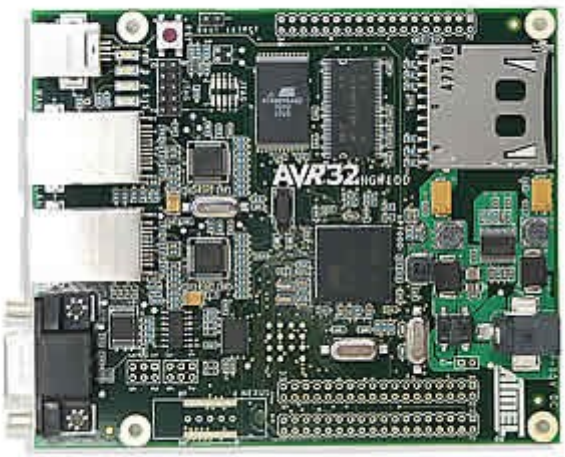


Abbildung 50: NGW100 als Plattform für die Management-Konsole

Als Basis für die Installation von Python auf dem NGW100 dienten diverse Ressourcen [25] aus dem Internet.

Die Ressourcen für die Implementierung befinden sich auf der CD.

6. Implementierungen Peripherie

In diesem Kapitel werden ein paar über die Adapter-Box angesteuerte Beispielgeräte technisch beschrieben. Es existieren noch weitere Versuchsaufbauten und Geräte, deren Erwähnung und Beschreibung den Rahmen der Masterarbeit aber überschreiten würden. Deshalb wurden nur folgende Geräte ausgewählt und erläutert.

6.1. Waage

Digitale Waagen, die per RS232 angeschlossen werden gibt es in vielen Varianten. Die hier vorgestellte Anbindung ist mit der Waage PCE TS 150 (siehe Abbildung 51) implementiert worden. Auf der RS232-Schnittstelle wird mit der entsprechenden Baudrate, die über das Display eingestellt werden kann sekündlich das aktuelle Gewicht ausgeben.

Abbildung 51: PCE TS 150 (Bild vom Hersteller)

Markiert ist der aktuelle Wert mit 0x0203 als Starttoken. Anschließend folgen 7 Werte, welche als ASCII-Folge den Wert, der auch auf dem Display sichtbar ist, ausgegeben.

```
(1)       int found_start=0;
(2)
(3)       int mindestens_werte = 0;
(4)         while(1){
(5)             res = read(fd,buf,1);
(6)             for(i=0;i<res;i++){
(7)                 puffer[index++] = buf[i];
(8)             }
(9)             if(buf[0]==0x02)
(10)            {
(11)                    found_start=1;
(12)                    index=0;
(13)            }
(14)
(15)            if(buf[0]==0x03 && found_start)
(16)            {
(17)                    mindestens_werte ++;
(18)
(19)                    if(mindestens_werte > 10)
(20)                    {
(21)                    for(i=0;i<7;i++)
(22)                            printf("%c",puffer[i+4]);
(23)                    printf("\n");
(24)                    break;
(25)                    } else
(26)                    {
(27)                            found_start=0;
(28)
(29)                    }
(30)            }
(31)            buf[res]=0;
(32)        }
```

Mit einem Aufruf der Ressource Gewicht wird ein ein neuer Wert von der Waage eingelesen und entsprechend zurückgegeben.

6.2. Barcode-Scanner

USB-Barcode-Scanner gehören zur Gruppe der HID Geräteklasse. Mittels Report-Descriptoren geben sie Ihre Daten an den Host weiter. Das Wesentliche für den Empfang der HID-Report-Desciptoren ist der Task host_mouse_hid_task:

Abbildung 52: Barcode-Scanner

```
(1)     //!
(2)     //! @brief This function manages the host mouse HID task.
(3)     //!
(4)     #ifdef FREERTOS_USED
(5)     void host_mouse_hid_task(void *pvParameters)
(6)     #else
(7)     void host_mouse_hid_task(void)
(8)     #endif
(9)     {
(10)      U8 i;
(11)
(12)    #ifdef FREERTOS_USED
(13)      portTickType xLastWakeTime;
(14)
(15)      xLastWakeTime = xTaskGetTickCount();
(16)      while (TRUE)
(17)      {
(18)        vTaskDelayUntil(&xLastWakeTime,
   configTSK_USB_HHID_MOUSE_PERIOD);
(19)
(20)    #endif  // FREERTOS_USED
(21)        // First, check the host controller is in full
   operating mode with the
(22)        // B-device attached and enumerated
(23)        if (Is_host_ready())
(24)        {
(25)          // New device connection (executed only once after
   device connection)
(26)          if (mouse_hid_new_device_connected)
(27)          {
(28)            mouse_hid_new_device_connected = FALSE;
(29)
(30)            // For all supported interfaces
(31)            for (i = 0; i < Get_nb_supported_interface(); i++)
(32)            {
(33)              if(Get_class(i)==HID_CLASS &&
   Get_protocol(i)==MOUSE_PROTOCOL)
(34)              {
```

```
(35)                host_hid_set_idle(HID_IDLE_DURATION_INDEFINITE,
   HID_REPORT_ID_ALL, i);
(36)                host_hid_get_report(HID_REPORT_DESCRIPTOR, 0,
   i);
(37)                pipe_mouse_in = Get_ep_pipe(i, 0);
(38)                Host_enable_continuous_in_mode(pipe_mouse_in);
(39)                Host_unfreeze_pipe(pipe_mouse_in);
(40)                mouse_hid_connected=TRUE;
(41)                break;
(42)              }
(43)            }
(44)          }
(45)
(46)          if( Is_host_mouse_hid_configured() )
(47)          {
(48)            if((Is_host_in_received(pipe_mouse_in)) &&
   (Is_host_stall(pipe_mouse_in)==FALSE) )
(49)            {
(50)               Host_reset_pipe_fifo_access(pipe_mouse_in);
(51)               usb_report[0]=
(52)               usb_report[1]=
(53)               usb_report[2]=
(54)               usb_report[3]=0;
(55)               host_read_p_rxpacket(pipe_mouse_in,
   (void*)usb_report, 4, NULL);
(56)               Host_ack_in_received(pipe_mouse_in);
(57)               Host_free_in(pipe_mouse_in);
(58)    ….........AddScan(usb_report);
(59)            }
(60)            if(Is_host_nak_received(pipe_mouse_in))
(61)            {
(62)               Host_ack_nak_received(pipe_mouse_in);
(63)            }
(64)          }
(65)        }
(66)
(67)
(68)    #ifdef FREERTOS_USED
(69)      }
(70)    #endif
(71)    }
```

Nach der Erkennung des USB-Gerätes werden anliegende Report-Deskriptoren mit der Funktion AddScan der internen Queue beigefügt. Abhängig vom Modus werden beim Zugriff über REST nur der letzte oder alle noch nicht ausgegebenen Barcodes zurückgegeben.

6.3. RFID Leser für Anmeldung

Schnell wurde festgestellt, das ein häufiges Wechseln zwischen den Stationen beim An- und Abmelden ein technisches Hilfsmittel benötigt. Als Lösung wurde ein freier RFID-Leser gefunden, über den man sich mittels einer RFID-Karte schnell an- und abmelden kann. Der Aufbau befindet sich noch auf dem Labortisch, demonstriert jedoch bereits die Grundfunktion.

Anstelle von Benutzername und Passwort wird ein mobiles RFID-Tag in die Nähe des Lesers der Station geführt. Über einen AJAX-Mechanismus (Polling auf REST-Kompomente) im Anmeldefenster wird der Benutzer am Leser erkannt und es wird die Standardoberfläche geladen, bzw. der Benutzer wird abgemeldet, sofern er sich zuvor bereits angemeldet hat.

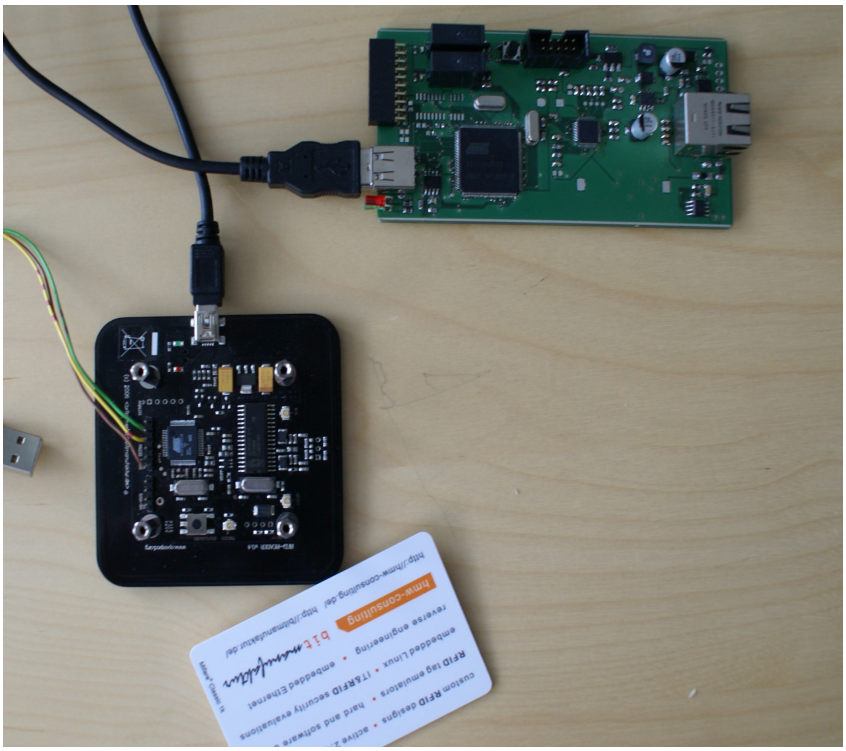


Abbildung 53: OpenPCD

6.4. URL-Button

Mit dem URL-Button kann per Knopfruck eine beliebige Adresse aus dem Internet aufgerufen werden. In der Adapter-Box sind alle Mechanismen hierfür vorhanden. Per GPIO (Digitaler Ein- und Ausgang) wird ein Taster angeschlossen.

Der Aufruf erfolgt über eine Anfrage der URL auf die gleiche Art und Weise, wie bei einer asynchronen REST-Antwort. Über die Adapter-Box kann ebenfalls der Zustand des Knopfes per Polling beobachtet werden, wobei der asynchrone Zugriff wesentlich Ressourcen schonender ist.

Einsatzmöglichkeiten für den URL-Button wäre z.B. die Meldung von bestimmten Ereignissen, wie: Der Postbote war da - jetzt kann eine E-Mail an jeden Kunden versendet werden - „Ihre Bestellung wurde unserem Versandunternehmen übergeben.“

6.5. USB-API

Die USB-API ermöglicht ohne Eingriff in den Quelltext der Adapter-Box den Zugriff auf USB-Geräte. Viele USB-Geräte arbeiten mit einfachem Control- Bulk und Interrupt-Transfer. Typischerweise sind das Geräte, die mit der Bibliothek libusb aus dem Benutzerbereich eines Betriebssystems (ohne Entwicklung eine Kerneltreibers) genutzt werden können.

Die Enumeration erfolgt zentral im USB-Stack. Der Zustand der Enumeration kann dem Request /usbinfo entnommen werden.

Um über einen Endpunkt Kommunizieren zu können, muss dies Adresse mir der Funktion Get_ep_pipe(i, 1) unter Angabe der Endpunktnummer ermittelt werden. Der Rückgabewert kann als Pipe für die Kommunikation in die Funktion host_send_data eingefügt werden.

host_send_data(g_pipe_ms_out, sizeof(buf_cmd), buf_cmd);

Zusätzlich wird noch der Speicher und die Länge des zu übertragenden Inhalts angegeben.

```
(1)     Host_freeze_pipe(pipe_cdc_data_bulkin);
(2)     if (PIPE_GOOD == host_send_data(pipe_cdc_data_bulkout,
             host_tx_cnt, cdc_stream_out_array))
(3)     host_ex_cnt = 0;    // if frame not sent, will try again
   next time (no data loss)
```

```
(4)     Host_unfreeze_pipe(pipe_cdc_data_bulkin);
```

Daten vom USB können mit der Funktion host_read_pipe_data gelesen werden.

```
(1)     while( (host_rx_cnt != STREAM_IN_SIZE) &&
        (Host_byte_count(pipe_data_bulkin) != 0) )
(2)     {
(3)          stream_in_array[host_rx_cnt] =
(4)               Host_read_pipe_data(pipe_data_bulkin, 8);
(5)          host_rx_cnt++;
(6)          LED_Toggle(LED0);
(7)     }
(8)     if( Host_byte_count(pipe_data_bulkin) == 0 )
(9)     {
(10)         Host_ack_in_received(pipe_data_bulkin);// pipe is empty
(11)         Host_free_in(pipe_data_bulkin);// ready to receive more
(12)    }
```

Per REST kann wie in Tabelle 13 dargestellt zugegriffen werden.

Beschreibung	Ressource	Methode
USB Bulk Transfer TX	/usb/bulk/write/{ep}?value={wert}	POST
USB Bulk Transfer RX	/usb/bulk/read/{ep}	GET
USB Contrrol Transfer	/usb/control?value={value}	GET,POST

Tabelle 13: USB-Kommandos

Der Zugriff über die USB-API erlaubt eine einfache „USB over LAN“ Anbindung. Für viele unkomplizierte USB-Geräte mag diese Anbindung ausreichen. Der Entwickler der Anwendung muss jedoch Kenntnisse über die USB-Kommunikation haben.

6.6. LCD-Einheit

Bringt man ein Display und zwei bis drei Buttons auf der Adapter-Box an, kann mit minimalem Aufwand ein „Pager“-ähnliches Gerät aufgebaut werden. Aus der Webanwendung kann schnell eine Nachricht auf das Display gesendet werden. Als Kommissionierungshilfe oder Nachrichten-Box kann diese Schaltung für diverse Prozesse eine Erleichterung bedeuten. Die Taster werden intern für ein eigenes Menü abgefragt oder können über den URL-Button-Mechanismus in die Anwendung integriert werden.

Abbildung 54: 4x20 Zeichendisplay für Nachrichten

Des weiteren muss ein Batteriebetrieb bzw. Akkubetrieb möglich sein. Die Adapter-Box benötigt über 5V, welche man einfach aus 4 Mignon AA Zellen entnehmen kann.

6.7. USB-Steckdose für Power-Management

Für bereits unter 50 EUR gibt es im Elektro-Versandhandel USB-Steckdosen, die es ermöglichen die Steckplätze in der Stromleiste mittels Programm gezielt ein- und auszuschalten. Mittels einem USB-Sniffer wurde das Protokoll analysiert und entsprechend in der Adapter-Box implementiert.

Erkannt werden die Stromleisten als HID-Implementierung (wie Barcode-Scanner). Über Standard-Descirptoren können die Dosen ein- und ausgeschaltet werden. Die Werte können der Tabelle 14 entnommen werden.

Dose	Dose einschalten	Dose ausschalten
1	0x0101	0x0100
2	0x0301	0x0300
3	0x0601	0x0600
4	0x0901	0x0900
5	0x0A01	0x0A01

Tabelle 14: Protokoll USB-Stromdose

Abbildung 55: USB-Steckdose von Versandhandel

7. Software API für PHP

Die Implementierungssprache des Warenwirtschaftssystems ist PHP. Im Folgenden wird daher die Anbindung an diese Programmiersprache gezeigt. Die Integration der REST-Schnittstelle in andere Programmiersprachen geschieht im Wesentlichen auf dem selben Weg. Mit Hilfe der Funktionen der PHP-Bibliothek können Geräte direkt als PHP-Objekte verwendet werden. Die Bibliotheken „mapped“ die Aufrufe auf die Hardware so, dass es für den Webentwickler genauso wie ein integrierter Datenbankzugriff oder eine einfache Dateioperation aussieht.

7.1. Funktionsübersicht

Für die Kommunikation werden Funktionen zum Verbindungsaufbau, den Zugriff auf die Hardware-Ressourcen und Dienste der Management-Konsole benötigt. Im Folgenden werden die wichtigsten und aktuell implementierten Funktionen aufgelistet und beschrieben.

Verbindung zu Management-Konsole öffnen

Mit dem Anlegen eines Objektes der Klasse ABConnection wird eine Verbindung zu der Management-Konsole aufgebaut. Als Parameter muss die aktuelle IP-Adresse der Konsole angeben werden. Standardmäßig ist diese 10.0.0.1.

```
(1)     $tmp = new ABConnection("10.0.0.1");
```

Nach dem erfolgreichen Öffnen der Verbindung kann mit dem erstellten Objekt auf weitere Funktionen zugegriffen werden.

Ausgabe der aktuell Teilnehmerliste des Netzwerks

Eine Liste mit allen am Netzwerk registrierten Teilnehmern liefert die Funktion GetDevices(). Als Ergebnis wird ein assoziatives Array ausgegeben.

```
$tmp->GetDevices();
```

IP für Label ermitteln

Für die direkte Kommunikation mit einer Adapter-Box wird ihre IP-Adresse benötigt. Mit der Funktion GetIPLabel kann eine IP-Adresse zu einem gegebenen Label ermittelt werden. Als Rückgabewert erhält man die IP-Adresse als String. Ist keine IP-Adresse für das Label vorhanden wird ein leerer String gesendet.

```
(2)     $tmp->GetIPLabel("waage01");
```

Nachricht an Teilnehmer direkt senden

Mit der durch GetIPLabel ermittelten IP-Adresse kann direkt ein Aufruf an eine Adapter-Box erfolgen.

```
(3)     $tmp->CommandDirectIP("192.168.0.104",“gewicht“);
```

Beide Schritte werden in der Funktion CommandDirect vereint. Intern wird erst die IP-Adresse ermittelt, um schließlich das Kommando direkt an das Gerät senden zu können.

```
(4)     $tmp->CommandDirect("waage01",“gewicht“);
```

Management-Konsole als Gateway-Interface verwenden

Alternativ zur direkten Kommunikation kann ein Aufruf über die Management-Konsole versendet werden. In diesem Fall ruft die Management-Konsole intern die entsprechende Ressource des angegeben Gerätes an und gibt das Ergebnis an die ursprüngliche Anfrage zurück. In der Software kann so ohne vorheriges auflösen des Labels in eine IP-Adresse auf eine Ressource zugegriffen werden.

```
(5)     $tmp->Command("waage01",“gewicht“); // Gewicht abholen
(6)     $tmp->Command("versand01",“off“); // PC ausschalten
```

Computer oder Gerät ein- bzw. ausschalten

Eine spezielle Ressource ist das Ein -und Ausschalten jedes Teilnehmers des Netzwerkes. Das Kommando muss beim Einschalten immer über die Management-Konsole gesendet werden, denn nur diese kann das Gerät starten.

```
(7)     $tmp->Wakeup("bene-laptop");
```

Der Einfachheit halber bietet die Bibliothek auch die Funktion Shutdown an. Intern wird diese als direktes Kommando gehandhabt, jedoch ist der Einsatz für den Entwickler einfacher.

```
(8)     $tmp->Shutdown("bene-laptop");
```

Die Bibliothek bietet die Basis für eine höhere Schicht, welche die einzelnen Geräte als Objekte abbildet. Mehr dazu im Absatz 7.3. Objekt-Mapping.

7.2. REST-Zugriff

Der Zugriff auf eine REST-Komponente wird auf zwei Wegen realisiert. Der Einfache baut ein Socket auf und tauscht HTTP-Nachrichten folgendermaßen aus:

```
(1)       function SocketSend($ip,$port,$url,$answer=1)
(2)       {
(3)         $ret = "";
(4)         $fp = fsockopen($ip, $port, $errno, $errstr, 30);
(5)         if (!$fp) {
(6)           $ret = "$errstr ($errno)<br />\n";
(7)         } else {
(8)           $out = "GET $url HTTP/1.1\r\n";
(9)           $out .= "Host: $ip\r\n";
(10)          $out .= "Connection: Close\r\n\r\n";
(11)          fwrite($fp, $out);
(12)          if($answer==0)
(13)            return;
(14)        do {
(15)          $line = fgets($fp);
(16)
(17)          if ($line === false)
(18)            break;
(19)          $ret .= trim($line);
(20)
(21)          if(strpos($line,"END"))
(22)            break;
(23)        } while(true);
(24)
(25)          fclose($fp);
(26)        }
(27)        return $ret;
(28)      }
```

Eine Implementierung mit der Funktion „SocketSend“ würde wie folgt aussehen:

```
(1)       function GetIPLabel($label)
(2)       {
(3)         $result = $this->SocketSend($this->management_konsole,
             8080,"/devices");
(4)         $result = $this->DecodeMessage($result);
(5)
(6)
(7)         $rows = split("!",$result);
(8)         for($i=0;$i<count($rows);$i++)
(9)         {
(10)          $device = split(";",$rows[$i]);
(11)          if($device[0]==$label)
(12)            return $device[2];
(13)        }
(14)        return -1;
(15)      }
```

An die Management-Konsole wird der REST-Aufruf /devices per GET gesendet, welcher als Antwort eine Liste mit den Teilnehmern liefert. DecodeMessage extrahiert den Inhalt

der Nachricht. Mittels einer Schleife wird nach dem passenden Label gesucht, und bei einem Treffer die IP-Adresse entsprechend zurückgegeben.

Der aufwändigere Weg der Implementierung des Zugriffes auf eine REST-Komponente ist der, bei welchem ein PHP HTTP-Client (http://scripts.incutio.com/httpclient/) als Basis dient. Der Client unterstützt alle wichtigen Mechanismen wie z.B. POST, Upload, Redirect, usw.:

```
(1)     $client = new HttpClient('www.amazon.com');
(2)     $client->setDebug(true);
(3)     if (!$client->get('/')) {
(4)         echo '<p>Request failed!</p>';
(5)     } else {
(6)         echo '<p>Amazon home page is '.strlen($client-
   >getContent()).' bytes.</p>';
(7)     }
```

7.3. Objekt-Mapping

Für die implementierten Geräte des Netzwerkes gibt es „Wrapper“, welche die Hardware für den Programmierer noch weiter abstrahieren. Die Klassen befinden sich auf der CD im Anhang.

So kann beispielsweise auf eine Waage einfach mittels $waage->GetGewicht(), oder einem Barcode-Scanner mittels $barcode->GetScan() zugegriffen werden.

Bevor der Zugriff mit den hier vorgestellten Klassen funktioniert, muss jedoch zuvor einmal zentral auf die Management-Konsole zugegriffen werden. Über eine Session-Variable wird die IP-Adresse hinterlegt und steht nun für weitere Aufrufe zur Verfügung.

```
(1)     $tmp = new ABConnection("192.168.0.107");
```

Im folgenden sind die wichtigsten Klassen für das Warenwirtschaftssystem beschrieben:

Zugriff auf eine Waage

Unter Angabe des Labels kann mit der Methode GetGewicht, das aktuell vorliegende Gewicht des Artikels auf der Waage abgeholt werden.

```
(1)     $myWaage = new Waage(„waage01“);
(2)     echo $myWaage->GetGewicht();
```

Zugriff auf einen Barcode-Scanner

Der Barcode-Scanner kennt zwei Betriebsarten. Zum einen den „Single-Shot“ Modus – in welchem immer nur ein Scan einzeln aufgenommen wird und bereit liegt. Zum anderen den „Continous-Mode“ Modus – in dem viele Scans gesammelt und als Lite weiterverarbeitet werden. Abhängig von dem Arbeitsgang ob z.B. Artikel einzeln eingelagert (Single-Shot) oder bei einer Inventur zu mehreren erfasst (Continous-Mode) und im Bulk verarbeitet werden muss der richtige Modus gewählt werden.

```
(1)     $barcode = new Barcode(„barcode01“);
(2)     echo $barcode->GetScan();
(3)     echo $barcode->GetScans();
```

Zugriff auf RFID-Leser für Anmeldung

Für den Zugriff auf den RFID-Leser - speziell für die Anmeldeseite - existiert ebenfalls ein Wrapper. Auf die genaue Implementierung des RFID-Lesers als Gerät wird in Kapitel 6.3 eingegangen.

```
(1)     $rfid = new RFIDLeser(„rfid01“);
(2)     $rfid->PollUser();
```

Jederzeit können weitere Klassen für neue Geräte basierend auf den vorliegenden Wrappern entwickelt werden.

8. Integration in das Warenwirtschaftssystem

Als eigenständige Entwicklung entstand im Auftrag der Firma embedded projects GmbH zuvor ein eigenes webbasierte Warenwirtschaftssystem. Das System wurde optimiert für die nahtlose Integration von Hardware in die Geschäftsabläufe. In diesem Kapitel soll als Beispielanwendung gezeigt werden, wie das Netzwerk der Masterarbeit in eine Anwendung integriert werden kann. Abbildung 56 zeigt das Anmeldefenster der Anwendung. Auf der Entwicklungsversion ist hier bereits der RFID-Leser-Mechanismus implementiert. Das Anmelden kann bequem mittels RFID-Tag passieren.

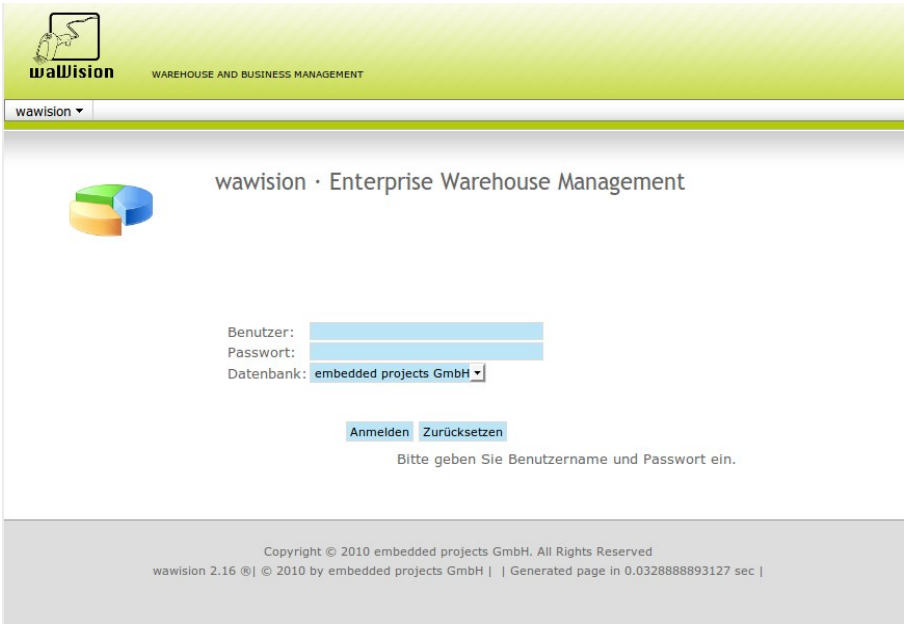


Abbildung 56: Anmelde Fenster für waWision

Der Rahmen der Anwendung basiert auf dem Framework phpWebFrame. Das Framework ist eine Eigenentwicklung. Wesentliche Vorteile sind:

- Es gibt einen Python Formulargenerator der aus HTML-Seiten fertige Formulare erzeugt. Mit ihm können Daten angelegt, bearbeitet und gelöscht werden.
- Integriertes Sicherheitskonzept (alle Daten werden nach Injections untersucht, bevor sie vom Entwickler bzw. der Anwendung genutzt werden können.)
- Einfache Aufteilung in Module

- Zentrales Benutzermanagement
- Methoden aus Modulen sind extern aufrufbar (z.B. über internen Prozessstarter)
- Einfaches Templatesystem für strikte Trennung zwischen Code und Layout
- Vollständige Integration der Yahoo (YUI) AJAX-Bibliothek
- Automatisches Logging der Operationen (für Analysen)
- Integrierter Profiler-Prozess für Optimierung von Engpässen

Abbildung 57 zeigt die Architektur der Anwendung. Die Adapter-Box-Bibliothek ist als eigenständiges Modul auf die gleiche Art wie z.B. das Datenbankmodul integriert. So kann zentral auf die angeschlossene Hardware zugegriffen werden.

Im Folgenden werden kurz verschiedene Stationen mit Geräten und Oberflächen vorgestellt, um einen Einblick in die Integration zu bekommen.

8.1. Versandstation

Die Versandstation ist optimiert für einen reibungslosen und fehlerfreien Ablauf des Versandprozesses. Im Versandprozess werden täglich Bestellungen, die zuvor aus dem Lager gebucht worden (Barcodescanner) sind, an Kunden versendet. Aufgaben der Versandstation:

- Scannen aller Artikel zu einer Bestellung
- Drucken des Lieferscheins und der Rechnung zum aktuellen Auftrag
- Erzeugen einer Paketmarke
- Erstellen eines Fotos der zu versendenden Artikel
- Wiegen der Artikel als weitere Prüfsumme
- Scannen der Trackingnummer (zum Senden per E-Mail an den Kunden)

Den Aufbau der Versandstation kann Abbildung 58 entnommen werden. Links begonnen steht der Drucker für die Paketmarken. Ein weiterer Drucker (im Bild nicht zu sehen) liefert Rechnung und Lieferschein. Der Barcode-Scanner ist fest über dem Arbeitsplatz montiert. Mit diesem muss jeder Artikel einzeln gescannt werden. Erst wenn alle Artikel der aktuell zu bearbeitenden Lieferung erfasst worden sind, wird das Foto geschossen und das Gesamtgewicht ermittelt. Anschließend werden die Rechnung und der Lieferschein gedruckt und es erfolgt ein Wechsel zur Oberfläche für das Drucken der Versandpapiere.

Die Zugriffe auf die Hardware sind direkt in den Programmfluss eingebunden. Eine Rechnung wird im Versandprozess beispielsweise so gedruckt:

```
(1)     $Brief = new RechnungPDF(&$this->app);
(2)     $Brief->GetRechnung($rechnung);
(3)     $tmpfile = $Brief->displayTMP();
(4)     $this->app->printer->Drucken($druckercode,$tmpfile);
```

Soll ein Artikel eingescannt werden, wird dieser intern aus dem sogenannten Zwischenlager entfernt. (siehe folgenden Quelltext). Der Zugriff auf den Scanner erfolgt

über die API (Zeile 1). Der anschließende Teil, welcher den Artikel aus dem Zwischenlager bucht, ist wie gewohnt in PHP geschrieben. Eine klassische Anwendung würde die Artikelid über GET oder POST empfangen – hier wird sie direkt aus der Queue der Adapter-Box des Barcode-Scanners per REST-Aufruf abgeholt.

```
(1)     $artikelid = $this->app->AB->GetScan("barcode_versand");
(2)
(3)     $tmpgeliefert  = $this->app->DB->Select("SELECT geliefert
            FROM lieferschein_position WHERE
            lieferschein='$lieferschein' AND artikel='$artikelid'
            AND geliefert<menge LIMIT 1");
(4)
(5)     $tmpgeliefert++;
(6)
(7)     $this->app->DB->Update("UPDATE lieferschein_position SET
            geliefert='$tmpgeliefert' WHERE id='$pos' AND
            lieferschein='$lieferschein' AND artikel='$artikelid'
            LIMIT 1");
(8)
(9)     $tmpgeliefertauftrag  = $this->app->DB->Select("SELECT
            geliefert_menge FROM auftrag_position WHERE
            auftrag='$auftrag' AND artikel='$artikelid' AND
            geliefert_menge<menge LIMIT 1");
(10)    $tmpgeliefertauftrag++;
(11)
(12)    $this->app->DB->Update("UPDATE auftrag_position SET
            geliefert_menge='$tmpgeliefertauftrag' WHERE
            id='$posauftrag' AND auftrag='$auftragid' AND
            artikel='$artikelid' LIMIT 1");
(13)
(14)    $this->app->DB->Update("UPDATE auftrag_position SET
            geliefert='1' WHERE auftrag='$auftragid' AND
            artikel='$artikelid' AND geliefert_menge=menge AND
            geliefert!='1' LIMIT 1");
(15)    //echo "artikel kommt in lieferung vor";
(16)
(17)    //artikel aus zwischenlager nehmen!!!!
(18)    $zwischenlagerid = $this->app->DB->Select("SELECT id FROM
            zwischenlager WHERE objekt='lieferung' AND
            artikel='$artikelid' AND menge > 0 LIMIT 1");
(19)    $zwischenlagermenge = $this->app->DB->Select("SELECT menge
            FROM zwischenlager WHERE id='$zwischenlagerid' LIMIT
   1");
(20)    $zwischenlagermenge = $zwischenlagermenge -1;
(21)    $this->app->DB->Update("UPDATE zwischenlager SET
            menge='$zwischenlagermenge' WHERE id='$zwischenlagerid'
            LIMIT 1");
```

Abbildung 58: Versandstation

8.2. Wareneingang

Im Wareneingang werden die Pakete, die per Post geliefert werden direkt per Kamera und Waage erfasst (siehe Abbildung 59). Das Bild dient zum einen der Dokumentation des Prozesses bzw. dem einfacheren späteren Zuordnen vom Inhalt des Pakets, zum anderen kann dem Kunden (sofern es sich um eine Rücksendung handelt) eine E-Mail über den erfolgreichen Empfang des Pakets mit einem Foto und der Angabe des Gewichts gesendet werden (siehe Abbildung 60).

Abbildung 59: Warenannahme mit Waage und Kamera

Der Zugriff der Hardware ist wieder direkt in den Programmfluss integriert. Auf die Kamera und Waage wird über das entsprechende Label über die integrierte API zugegriffen. Das Bild wird intern direkt in das Dokumentenmanagement eingebucht und das Gewicht wird ein der Datenbank abgespeichert.

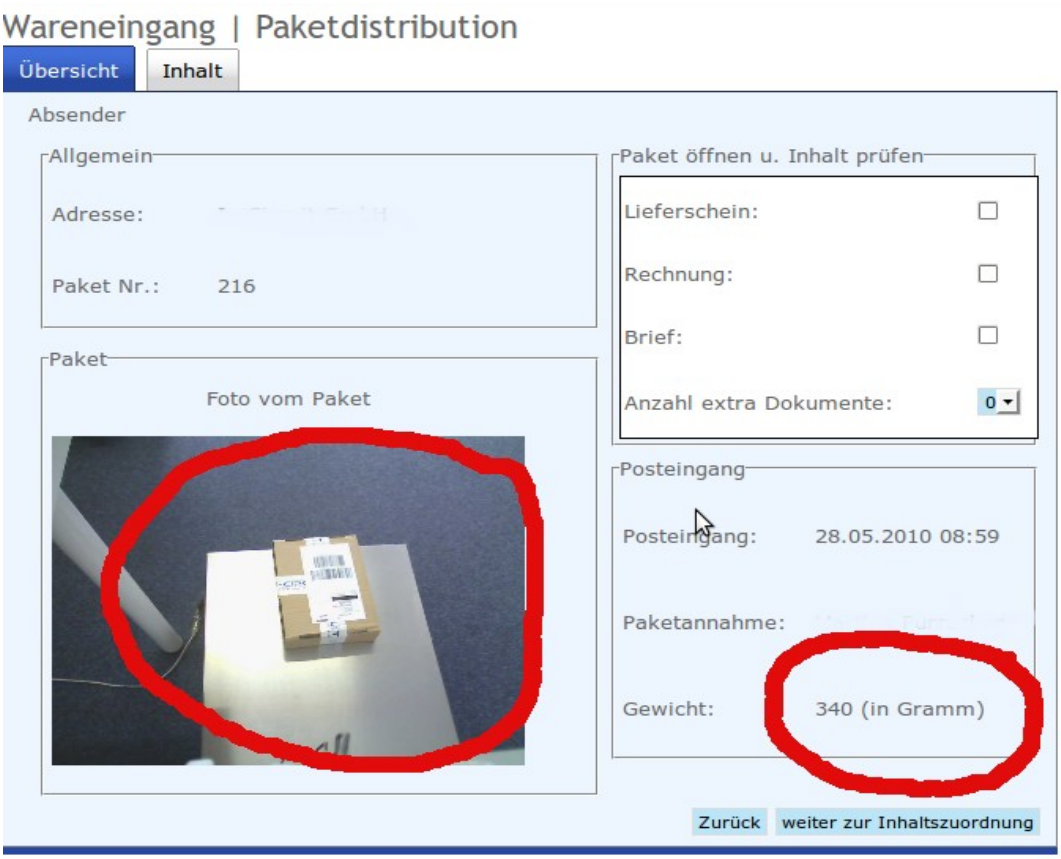


Abbildung 60: Paketannahme

8.3. Prozesse zur Automatisierung

Regelmäßig um 09:00 Uhr Werktags sollen die Stationen inkl. der angeschlossenen Peripherie eingeschaltet und um 17:00 Uhr - sofern kein Benutzer mehr eingeloggt ist - wieder ausgeschaltet werden. Im Warenwirtschaftssystem ist ein sogenannter Prozessstarter (siehe Abbildung 61) integriert. Der Prozessstarter wird typischerweise für Prozesse wie das Abholen der E-Mails von Kunden und Erstellen von Tickets, den Import von Aufträgen aus dem Online-Shop und eben auch für regelmäßige Zugriffe auf die Hardware verwendet.

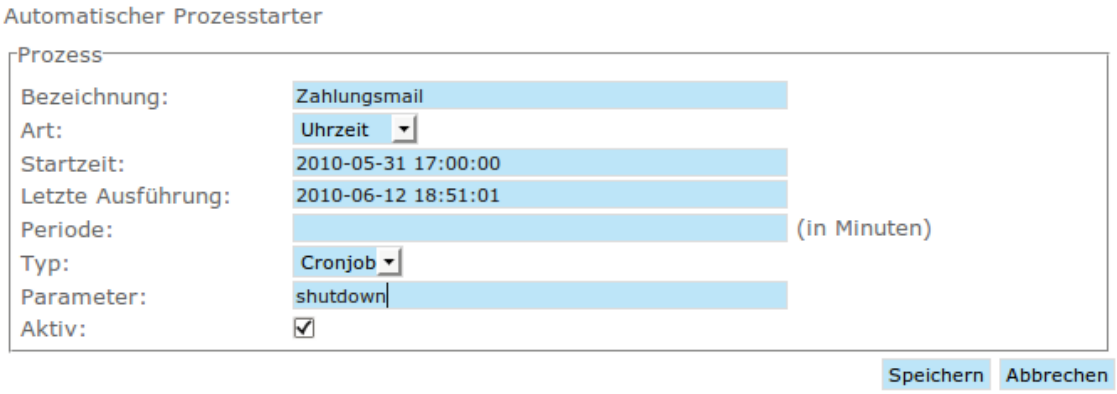


Abbildung 61: Prozessstarter

8.4. Management-Konsole Oberfläche

Abbildung 62 zeigt eine erste Oberfläche der Management-Konsole integriert in das Warenwirtschaftssystem. Stationen können einfach per Mausklick ein- und ausgeschaltet bzw. gezielt für den Prozessstarter konfiguriert werden. Die angezeigten Geräte werden über den Aufruf GetDevices() an der Management-Konsole ermittelt.

Abbildung 62: Management-Konsole

8.5. Lager mit Barcode-Scanner

Alle Artikel im Lager werden bei der Paketdistribution mit einem Barcode-Etikett versehen (siehe Abbildung 63).

Abbildung 63: Lager mit Barcode-Etiketten

Jede Regal-Zelle im Lager besitzt ebenfalls einen eigenen Barcode. Wird ein Artikel ein- oder ausgelagert, muss immer zuvor das Regal abgescannt werden, um die Daten in der Datenbank konsistent zu halten. In der Installation gib es im Moment zwei Etiketten-Drucker (für kleine und große Etiketten).

Mit dem Computer im Lager (Abbildung 64), an welchem sich ein Barcode-Scanner befindet, können die Waren ein- und ausgelagert werden. Noch ist der Scanner mit einem USB-Verlängerungskabel angeschlossen. In einem nächsten Schritt soll eine Adapter-Box mit Funkschnittstelle fest am Barcode-Scanner angebracht werden. Dadurch kann mit einem Handelsüblichen Barcode-Scanner eine einfache kostengünstige Funklösung realisiert werden. Auf diese Art können ebenfalls weitere USB-Geräte per Funk angebunden werden.

Abbildung 64: Lager Computer mit Barcode-Scanner

Schluss und Ausblick

Das Besondere und interessante an dieser Arbeit ist die Verbindung von Hardware über Webschnittstellen- bzw. Protokolle an Softwareanwendungen.

Im Einzelnen haben sich sich folgende Vorteile ergeben:

- Die Adapter-Box vereinfacht und abstrahiert den Zugriff auf die Hardware mittels einer einfachen Webschnittstelle, so dass Softwareentwickler ohne Hardwarekenntnisse Geräte selbst integrieren können.
- Weder die Installation noch die Konfiguration von Treibern ist notwendig.
- Die Anbindung verschiedenster Geräte durch die Vielfalt an unterstützen Schnittstellen (USB, RS232, RS485, IC2, etc.) erleichtert die Auswahl passender peripherer Komponenten.
- Die Ansteuerung jedes Geräts erfolgt zentral und unabhängig vom dem Computer, an dem es angebunden ist.
- Intelligentes Power-Management ermöglicht das Ein- und Ausschalten aller Rechner und Geräte von einem zentralen Punkt aus.
- Die gesamte Kommunikation findet ausschließlich über REST-Aufrufe statt, was bedeutet, dass es bereits einen großen Fundus an Software (Browser, Proxy, Netzwerk-Sniffer, Load-Balancer, u.v.m.) für Betrieb und Test gibt.
- Alle Geräte können sowohl bei Defekt als auch bei Umstrukturierung der Firma schnell ausgetauscht werden. Auch mobile Arbeitsplätze können rasch aufgebaut werden.
- Das gesamte Netzwerk steht unter der Open-Source Lizenz GPL.

In Verbindung mit Software kann die Adapter-Box einen echten Mehrwert für den Anwender schaffen. Vor allem in kleineren Firmen und Bereichen, in denen nicht so viele finanzielle Mittel zum Ausbau individuell angepasster Speziallösungen zur Verfügung

stehen, kann diese Methode sicher Punkten. Im täglichen Gebrauch hat sich die Ansteuerung externer Geräte mit der Adapter-Box in vielen Bereichen als sehr effizient erwiesen. Beispielsweise erleichterte die Erfassung aller Pakete mit Gewicht und Bild am Wareneingang die spätere Zuordnung aller Pakete bei der Warendistribution nach Inhalt und Aussehen enorm. Als besonders wichtig erwies sich die Anbindung externer Geräte mit der Adapter-Box bei Prozessen wie dem Versand. Im Lager werden alle Artikel per Scanner mittels Barecodeetiketten an Lager und Artikel ausgebucht. Der Versand erfolgt ebenso durch Barcodeerkennung der Artikeletiketten und Gewichtskontrolle. Dieses Verfahren ermöglicht einen nahezu hundertprozentig fehlerfreien Versand. Diese Quote hat sich vor allem im Praxistest positiv bestätigt. Als ebenso praktikabel erwies sich das zentrale Starten und Herunterfahren der Rechner und Geräte per Mausklick. Tägliche Kontrollen beim Verlassen des Büros wurden überflüssig und bei einigen Rechnern, die sonst oftmals tagelang im Dauerbetrieb waren, dürfte sich in Zukunft vielleicht auch die Lebensdauer erhöhen und der globale Stromverbrauch senken lassen.

Die Kombination zwischen einer Webanwendung und einer Anbindung von Embedded Systemen ist also äußerst spannend und vielseitig und eröffnet neue Ideen für andere Anwendungsbereiche. So könnte man Wetterstationen, Alarmanlagen, Haussteuerungen und viele weitere Anwendungen aus diesen Bereichen mit dieser Technik implementieren. Für die Zukunft ist demnächst eine weitere Überarbeitung der Adapter-Box geplant, die sie noch kompakter und kostengünstiger werden lässt.

Anhang

Foto der Adapter-Box

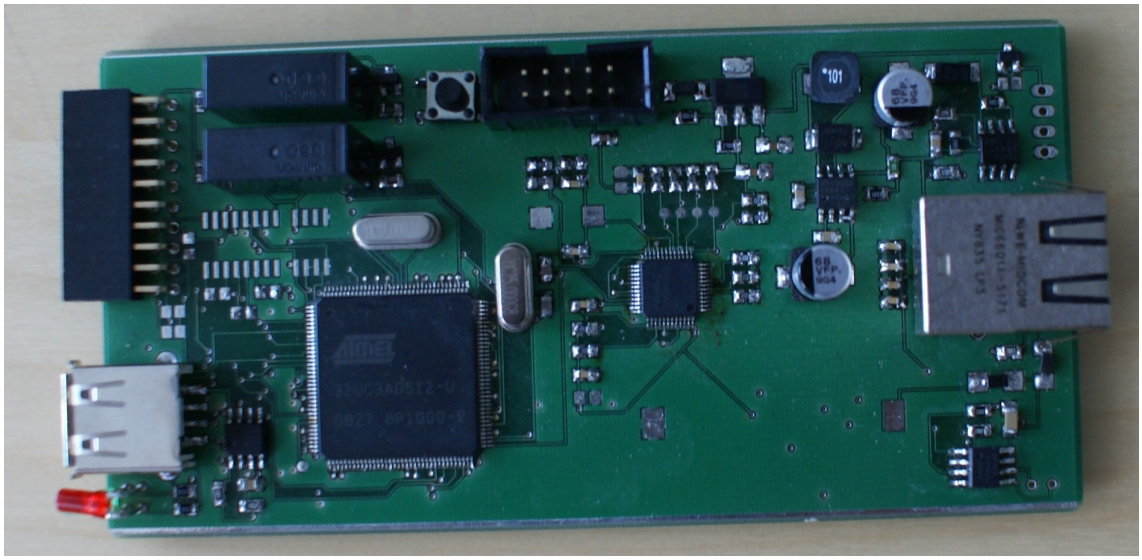


Abbildung 65: Bestückte Adapter-Box

Abbildung 66: Adapter-Box im Prototypgehäuse

Stückliste

(siehe CD)

Platinenlayout Top und Bottom

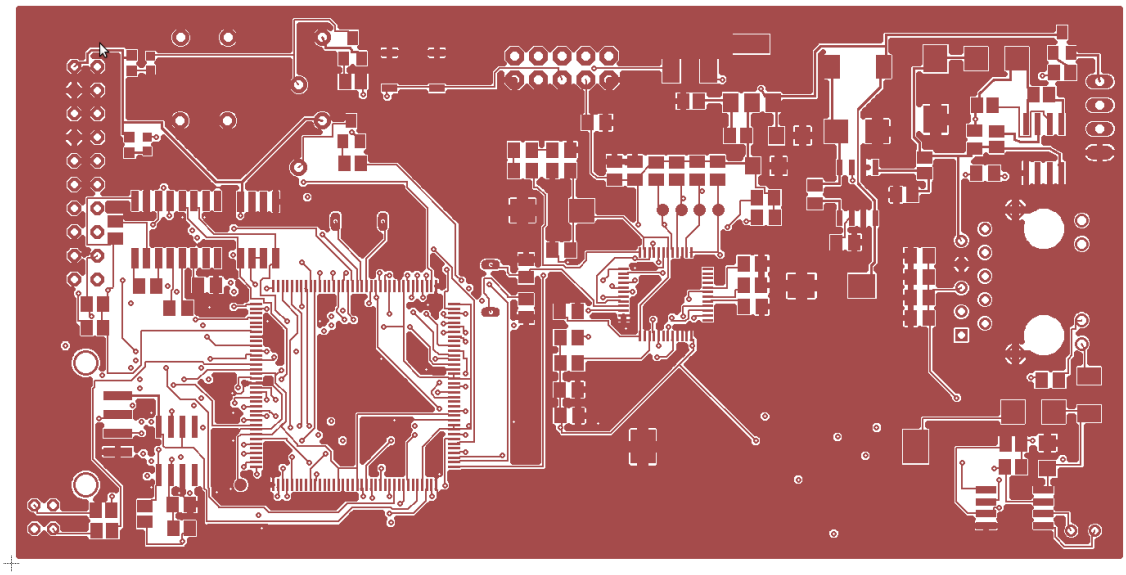


Abbildung 67: Layer TOP

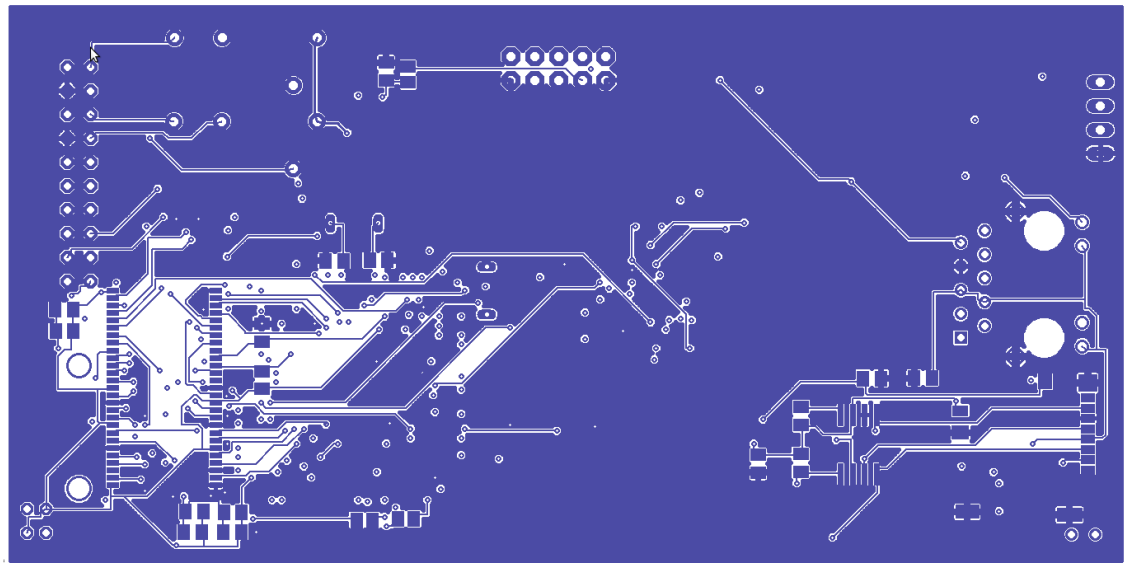
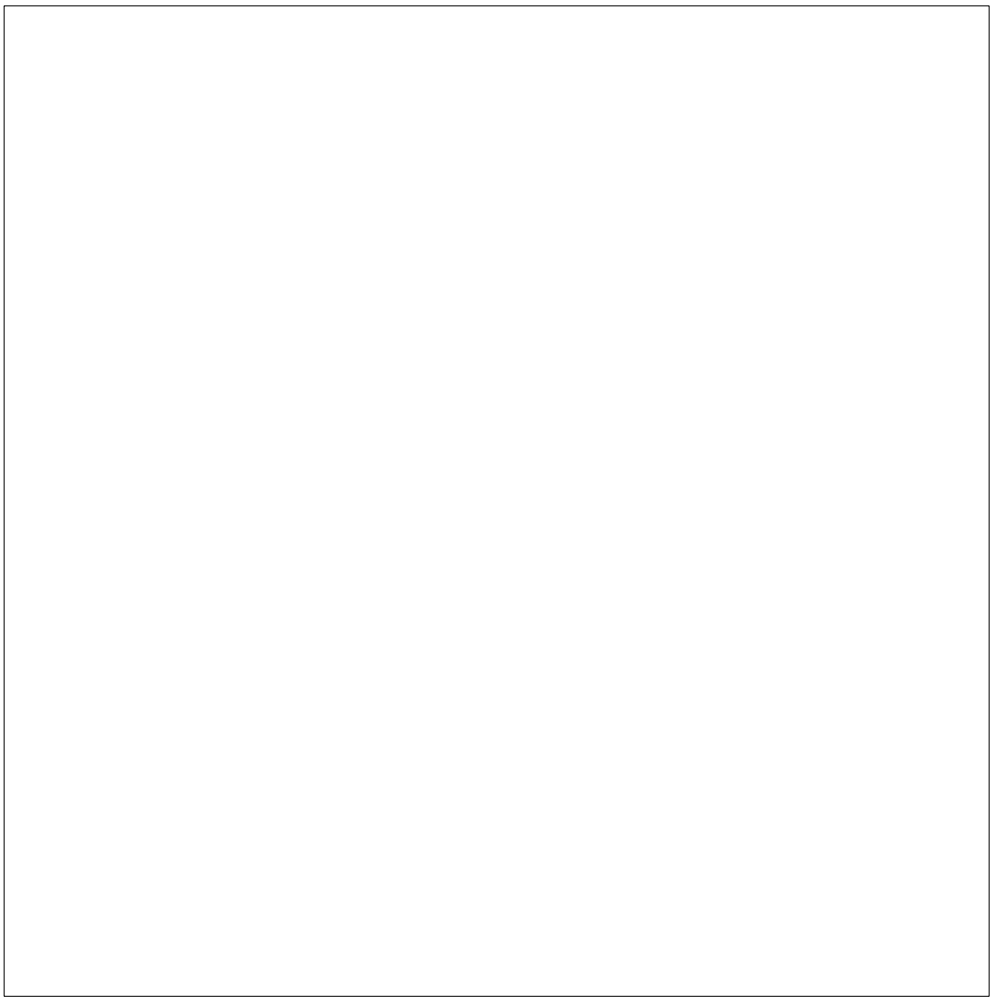


Abbildung 68: Layer BOTTOM

CD mit Quelltexten und Schaltungsunterlagen

- Stücklisten
- Schaltplan
- Eagle Schaltplan- und Layout-Dateien
- Quelltexte Adapter-Box
- Quelltexte Management-Konsole
- Anbindung Warenwirtschaftssystem



CD der Masterarbeit

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Geschäftsprozess, Wikipedia, 13.06.2010
http://de.wikipedia.org/wiki/Gesch%C3%A4ftsprozess

[2] Warehouse Managment, Michael ten Hompel, Thorsten Schmidt, Springer 2004

[3] Und es hat Pieps gemacht …, PHP Magazin 5.2009

[4] Service-orientierte Abbildung von Geschäftsprozessen mit freier Software,
Philipp Brune Linux Magazin 01/09

[5] Viper CE, http://www.aje.de/Bilder/Viper.pdf,13.06.2010

[6] Feldbusse, Wikipedia, 13.06.2010
http://de.wikipedia.org/wiki/Feldbusse

[5] REST Howto, 13.06.2010
http://wiki.linogistix.com/Wiki.jsp?page=LosProgrammersGuide#section-LosProgrammersGuide-SectionLosProgrammersGuideRemoteAndWebServices,

[7] Fraunhofer Institur Warehouse Management Internetseite, 2010
http://www.iml.fraunhofer.de/

[8] RFC 707,13.06.2010
http://tools.ietf.org/html/rfc707,

[9] W3C SOAP, 13.06.2010
http://www.w3.org/TR/SOAP/

[10] Microsoft, 13.06.2010
http://msdn.microsoft.com/de-de/library/ms809311.aspx

[11] OMG Group, 13.06.2010
http://www.omg.org/cgi-bin/doc?formal/04-03-12.pdf

[12] Fielding Roy Thomas: Architectural Styles and the Design of Network-bases Software Architectures. Doctoral dissertation, University of California, Irvine, 2000
http://www.ics.uci.edi/fielding/pubs/dissertation/top.htm

[13] Andrew D. Birrell, Bruce Jay Nelson; Xerox Palo Alto Research Center (Hrsg.),2010
http://www.cs.yale.edu/homes/arvind/cs422/doc/rpc.pdf

[14] Lingoistix, 2010
http://www.linogistix.com/de/index.html

[15] Warenwirtschaftssystem, Wikipedia, 13.06.2010
http://de.wikipedia.org/wiki/Warenwirtschaftssystem

[16] REST und HTTP, Stefan Tilkov, dpunkt.verlag 2009

[17] HTTP, Wikipedia, 13.06.2006
http://de.wikipedia.org/wiki/HTTP

[18] Datenblatt TPS2375, Texas Instruments TPS2375 SLVS525B – APRIL 2004 – REVISED APRIL 2008

[19] IEEE 802 LAN/MAN Standards Committee Mai 2005
http://grouper.ieee.org/groups/802

[20] Elektronik 26/2008, Weka Verlag, 2008

[21] OASIS Devices Profile for Web Services, 2009
http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01

[22] DPWS,Wikipedia 13.06.2010
http://de.wikipedia.org/wiki/DPWS

[23] Adam Dunkels, LwIP Network Stack, 13.06.2010
http://www.sics.se/~adam/

[24] AMD: Magic Packet Technology Whitepaper Rev. A (1995) (englisch, PDF)

[25] Building Python on the NGW100, 13.06.2010
http://www.cibomahto.com/2007/10/building-python-on-the-ngw100/