Fachhochschule Burgenland GmbH
Campus 1
A-7000 Eisenstadt

# Prototyp zur Nutzungsoptimierung von Photovoltaikenergie in Einfamilienhäusern

**Bachelorarbeit 2**

**zur Erlangung des akademischen Grades**

**Bachelor of Science in Engineering**

| | |
|---|---|
| Betreuer: | Dipl.-Ing. Dr. Robert Matzinger |
| Eingereicht von: | Ing. Andreas Vogler |
| Personenkennzeichen: | 1110640035 |
| Datum: | 15. Juni 2014 |

## Kurzfassung Deutsch

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung eines Prototypen, welcher durch Prozessleittechnik und smarter Steuerung den Eigenverbrauch von elektrischem Strom aus der Eigenproduktion von Photovoltaikanlagen in Einfamilienhäusern erhöht. Der Eigenverbrauch wird durch Automatisierung der Aufladung eines thermischen Speichers, in Abhängigkeit von produziertem Photovoltaikstrom, gesteuert.

Durch die zunehmende Verbreitung von stromproduzierenden Anlagen (vornehmlich Photovoltaikanlagen) nehmen die Vergütungen für in das Stromnetz eingespeisten Strom ab. Der Ertrag von eigenerzeugtem Strom liegt unter dem Preis von zugekauftem Strom. Durch die Erhöhung des Eigenverbrauches kann die Wirtschaftlichkeit einer Photovoltaikanlage entscheidend verbessert werden.

Aufbauend auf den Erkenntnissen der Bachelorarbeit 1 „Eigenverbrauchsoptimierung von Photovoltaikstrom in Einfamilienhäusern“ (Vogler, 2014), werden in der Arbeit die Teilkomponenten für ein derartiges Regelungs- und Steuerungssystem zusammengestellt und zu einem Gesamtsystem zusammengeführt. Ziel ist eine automatisierte Steuerung, damit der Eigenverbrauch erfolgt, wenn eigenproduzierter Strom im Überfluss vorhanden ist.

Aufgrund langer Produktzyklen erweisen sich Lieferanten im Steuerungs- und Regelungsbereich als vergleichsweise innovationsscheu und teuer. Eine Umsetzung mit industriellen Steuer- und Regelungstechnikbauteilen kam daher nicht in Frage. In dieser Arbeit wird gezeigt, wie eine Realisierung der geplanten Anlage mit Systemen wie Arduino und Raspberry Pi, die einen einfachen Einstieg in die Embedded Programmierung und Regelungs- und Steuertechnik ermöglichen, erfolgen kann.

Die Teilkomponenten wurden als eigenständige Komponenten umgesetzt, welche über Serviceschnittstellen mit dem Gesamtsystem kommunizieren. Diese lose Verbindung, durch die Verwendung von Services, führt zu einer hohen Verteilbarkeit der Teilsysteme, und folgt dem aktuellen IT-Trend von „IoT – Internet of Things“.

## Abstract

Within this thesis we implement a prototype of a smart home process control system for increasing the consumption of self-produced photovoltaic energy. To increase the consumption of self-produced energy, the energy is stored in a thermal storage by charging it automatically in case there is excess energy.

Due to the increasing popularity of small energy-producing plants (mainly photovoltaic systems) the allowances for energy fed into the public electricity grid decreases. The income from this kind of electricity is lower than the price of purchased electricity. By increasing the self-consumption of energy, the profitability of a photovoltaic system can be significantly improved.

Based on the findings of the thesis 1 „Eigenverbrauchsoptimierung von Photovoltaikstrom in Einfamilienhäusern“ (Vogler, 2014), the components to build such an integrated control system are implemented. The goal is to build an automated control system, that enforces power consumption when there is enough self-produced electricity available.

Due to long product cycles, companies in the process control-business are comparatively innovation averse and expensive. But with systems such as Arduino or Raspberry Pi, which allow an easy entry to the embedded programming and control technology, an intelligent control system for self-consumption of photovoltaic electricity by thermal storage can be implemented with low cost.

Our system consists of loosely coupled separated components. By the use of service interfaces, the components are highly distributed, which follows the current IT trend of " IoT - Internet of Things ".

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Der von Photovoltaikanlagen produzierte Strom kann in Einfamilienhäusern nicht immer unmittelbar bei Entstehung verbraucht werden. Damit ergibt sich ein Energieüberschuss, welcher in das öffentliche Netz eingespeist werden kann. Ist bislang die Einspeisung von selbst erzeugtem Solarstrom in das öffentliche Netz der Regelfall, so nimmt diese jedoch vor dem Hintergrund der sinkenden Einspeisevergütung an Bedeutung ab.

Das Speichern von elektrischer Energie in elektrischen Speicherzellen, wie Akkus, ist noch mit hohen Investitionskosten verbunden, und findet daher nur langsam Verbreitung in Eigenheimen. In Studien wird dem Eigenverbrauch eine große Bedeutung zugesprochen, da durch den Eigenverbrauch der Ertrag von Photovoltaikanlagen gesteigert werden kann, ohne dass in teure Speicherzellen investiert werden muss. Selbst genutzter Strom aus Eigenproduktion muss nicht unter dem Einkaufspreis an den Stromanbieter verkauft werden.

In einer Studie des IÖW (Institut für ökologische Wirtschaftsforschung) wird unter anderem hervorgehoben, dass smarte Regelungstechnik den Eigenverbrauch, in Abhängigkeit von der Photovoltaikstromerzeugung, wesentlich erhöhen kann. Jedoch wird auch darauf hingewiesen, dass die smarte Regelungstechnik derzeit noch zu teuer sei.

In der Bachelorarbeit 1 (Vogler, et al., 2014) wurde ein Lösungsweg zur Erstellung eines smarten Regelungssystem erarbeitet und passende Komponenten ausgewählt. Das dort beschriebene System bildet ein automatisiertes Regelungssystem zur Steuerung einer elektrischen Heizung eines thermischen Speichers. Stromüberschuss wird zum Heizen des Warmwassers im Eigenheim verwendet, wodurch einerseits ein Stromüberschuss selbst verbraucht wird, und andererseits der Energiebedarf aus anderen Quellen zum Aufheizen des Warmwassers verringert wird.

Die Motivation der vorliegenden Arbeit ist die Erstellung eines Prototypen, welcher die Funktionalitäten zur Erhöhung des Eigenverbrauches erfüllt, und der aufzeigt, dass smarte Regelungstechnik bereits mit kostengünstigen Komponenten, wie dem Raspberry Pi und dem Arduino Mikrokontroller, realisiert werden kann.

## 1.2 Zielsetzung und Fragestellung

Ziel dieser Arbeit ist es zu erforschen, ob der in der Bachelorarbeit 1 beschriebene Lösungsweg und die Systemarchitektur für eine Anlage, welche durch das automatisierte Steuern von Endgeräten (Stromverbrauchern) den Eigenverbrauch von selbst erzeugtem Strom erhöht, umsetzbar ist. Ein weiterer Aspekt der gewählten Lösungswege sind die Investitionskosten. Diese sollen gering gehalten werden. Es soll nach Möglichkeit ohne teure industrielle Anlagenkomponenten gearbeitet werden.

Nach den Erkenntnissen aus der Bachelorarbeit 1 soll eine automatisierte Steuerungslogik für das Aufheizen eines thermischen Warmwasserspeichers, umgesetzt werden. Die in der Bachelorarbeit 1 beschriebenen Lösungen für die Teilkomponenten werden evaluiert und jeweils eine der möglichen Lösungswege je Teilkomponente wird umgesetzt.

Im Zentrum des zu entwerfenden Systems steht die optimierte Steuerung, damit der Eigenverbrauch erfolgt, wenn eigenproduzierter Strom im Überfluss vorhanden ist.

Die Teilkomponenten sollen mit Komponenten wie Raspberry Pi und/oder Arduino umgesetzt werden. Software für die Datenverarbeitung sowie für die Steuerungs- und Regelungsaufgaben, soll auf Open-Source oder frei verfügbaren Softwareversionen aufbauen. Es kann jedoch nicht ausgeschlossen werden, dass auch kommerzielle Software zum Einsatz kommen wird.

In einer weiterführenden Arbeit könnte mit dem erstellten System der mögliche Ertrag und Nutzen des Systems, über die mit dem System erfassten Aufzeichnungen und Auswertungen, ermittelt werden.

## 1.3 Vorgangsweise und Methode

Im Sinne der Fragestellung wird in dieser Arbeit eine prottypische Anlage erstellt, welche mit smarter Regelung überschüssige Energie in einen thermischen Speicher überführt. Experimentelles Prototyping bietet den Vorteil, dass Lösungsansätze durch ein rasches Feedback auf deren Eignung untersucht werden können. Dadurch werden Probleme frühzeitig erkannt und ein Wechsel auf einen alternativen Lösungsansatz kann frühzeitig durchgeführt werden.

Die dafür benötigten Komponenten laut Forschungsplan (Vogler et al., 2014) werden als Teilkomponenten erarbeitet und zu einem Gesamtsystem zusammengeführt.

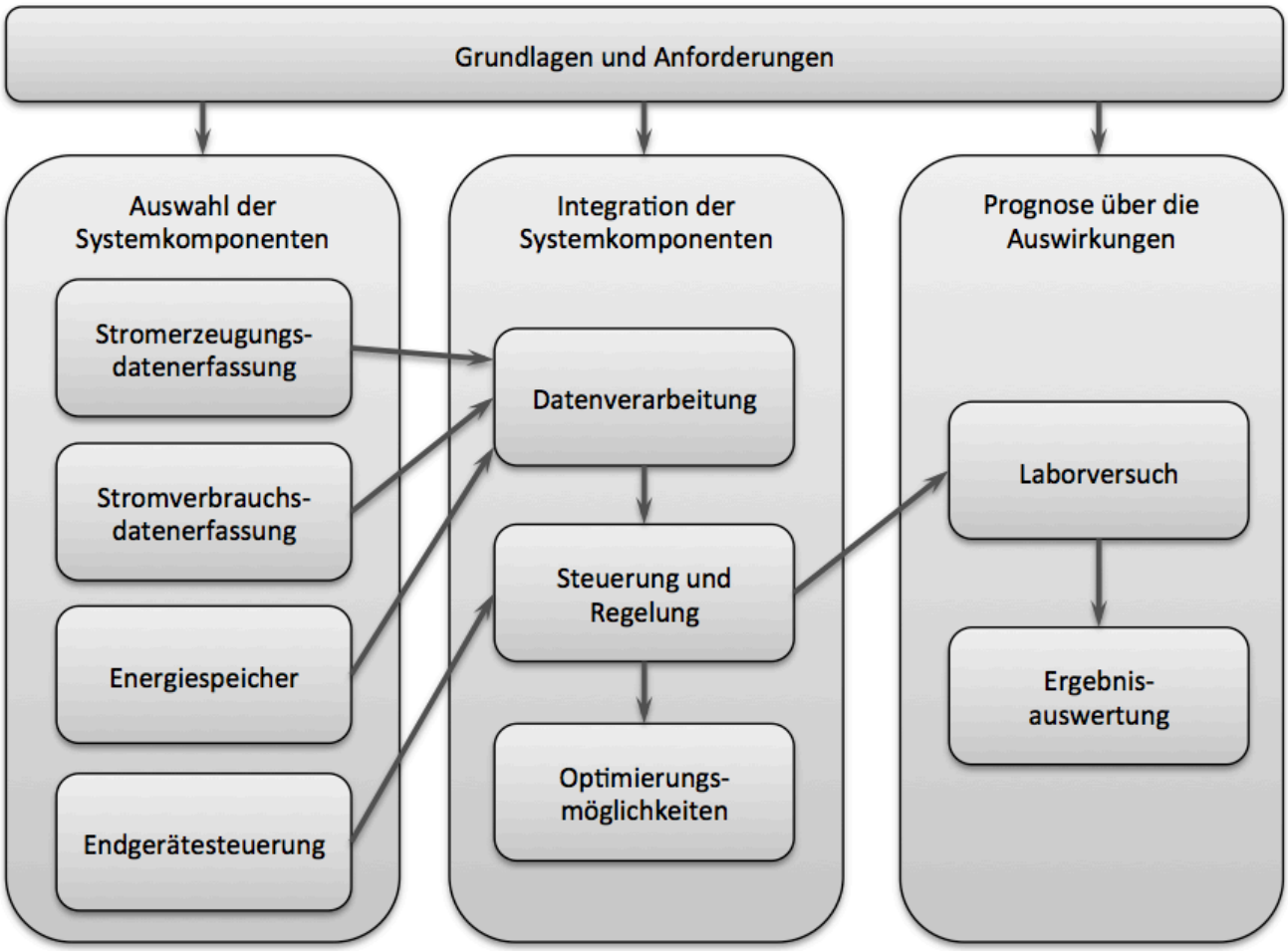



Abb. 1) Forschungsplan mit Abhängigkeiten der Komponenten (Vogler et al., 2014)

Folgende Aufgaben werden automatisiert und in ein Steuerungs- und Regelungssystem eingebunden:

- Erfassung der Stromerzeugung über Photovoltaikanlagendaten.
- Erfassung des Stromverbrauches im Haushalt über Stromzähler.
- Erkennung ob ein Überschuss an Energie vorhanden ist.
- Warmwasserboiler als Energiespeicher von Energieüberschuss und das Schalten eines Heizstabes zur Energieeinspeisung.
- Ermittlung und Erfassung der Regelgrößen mit welchen ermittelt wird zu welchem Zeitpunkt das Aufladen des Wärmespeichers erfolgen soll.
- Regelungs- und Steuerungssystem für das Aufladen des Wärmespeichers mit den ermittelten Daten als Regelgrößen.

# 2 Systemkomponenten

Die folgenden Kapitel beschäftigen sich mit den Systemkomponenten und wie diese in das Gesamtsystem, welches Energie aus Eigenproduktion intelligent in einem Energiespeicher (Warmwasserspeichers) speichert, eingebunden werden können.

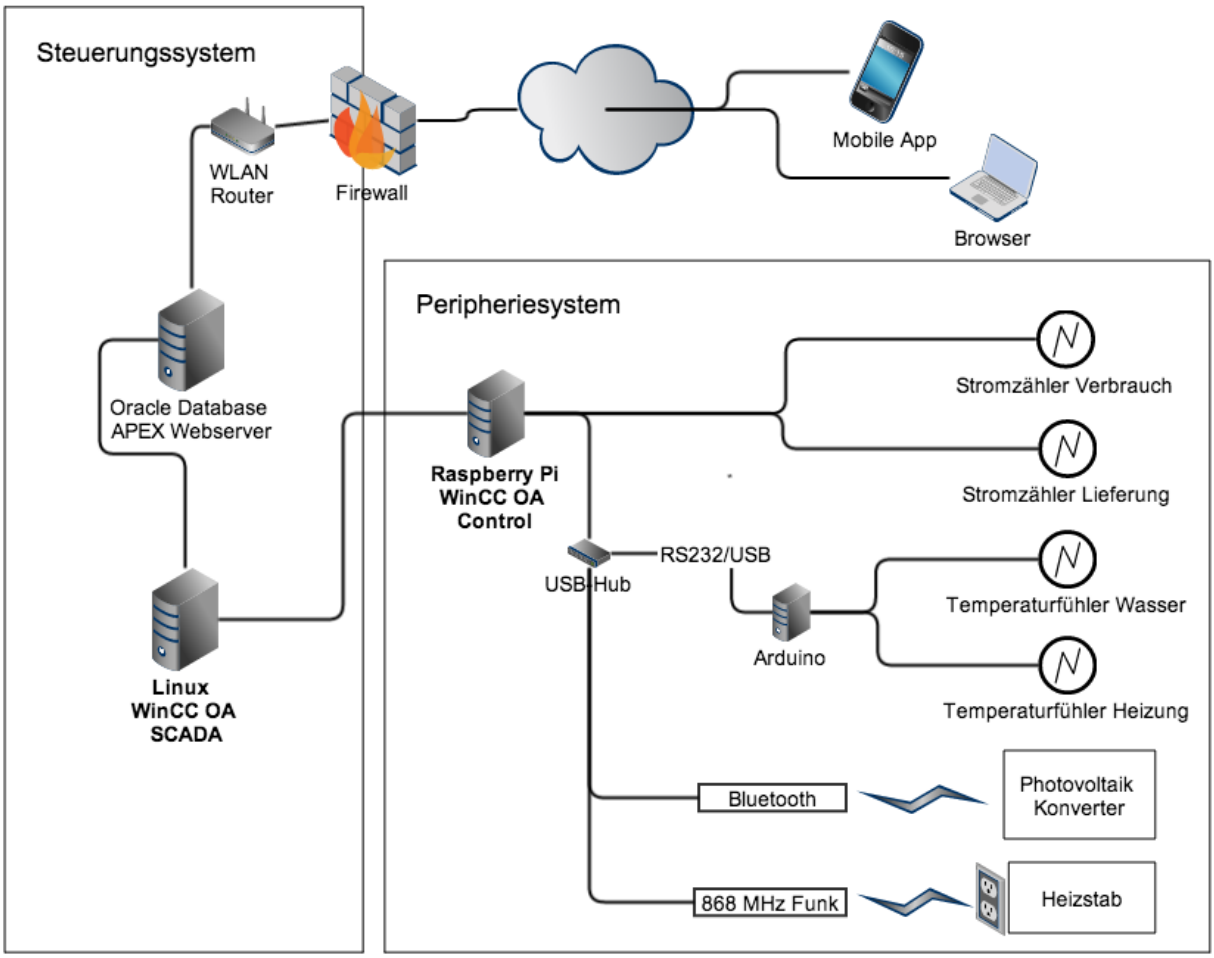



Abb. 2) Die Komponenten des Steuerungs- und Peripheriesystems

Das System besteht aus folgenden Komponenten:

- Erfassung der Stromerzeugung über Photovoltaikanlagendaten.
- Erfassung des Stromverbrauches im Haushalt über Stromzähler.
- Warmwasserboiler als thermischer Energiespeicher.
- Automatisiertes Schalten von Heizgeräten, welche zur Energieeinspeisung in den Wärmespeicher verwendet werden.

Die Aufgaben der Datenerfassung und der Endgerätesteuerung wird über das Peripheriesystem, welches aus einem Raspberry Pi sowie einem Arduino

Mikrokontroller besteht, durchgeführt. Diese befinden sich in der Nähe des Photovoltaikkonverters und des Warmwasserboilers.

Die übergeordnete Datenspeicherung und Verarbeitung sowie die Steuerung wird über das Steuerungssystem durchgeführt.

## 2.1 Softwarekomponenten des Systems

Die übergeordnete Datenspeicherung und Verarbeitung sowie die Steuerung wird über einen Steuerungs-PC durchgeführt. Auf diesem läuft das SCADA-System WinCC Open Architecture sowie eine Oracle Datenbank mit integriertem Webserver.

### 2.1.1 WinCC Open Architecture SCADA-System

Unter Supervisory Control and Data Acquisition (SCADA) versteht man das Überwachen und Steuern technischer Prozesse mittels eines Computersystems.

Automatisierungen werden, entsprechend der Automatisierungspyramide, in mehrere Schichten unterteilt.

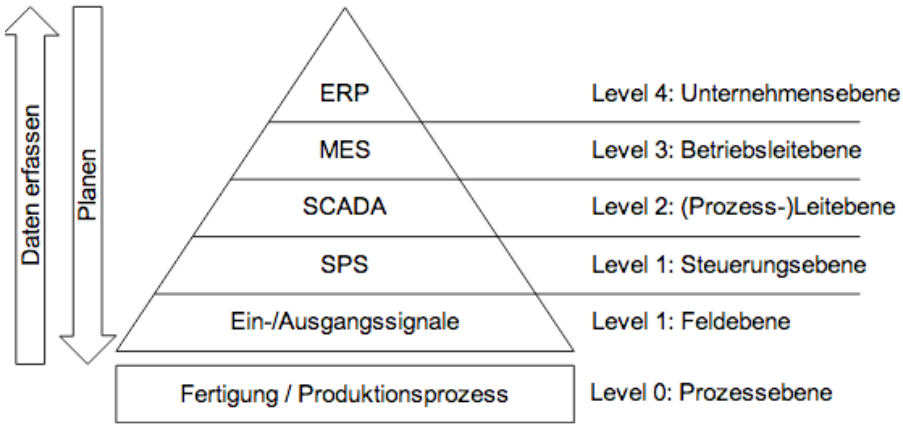



Abb. 3) Automatisierungspyramide

Dabei ist das Level 1 die prozessnahe Schicht, in welcher die Anbindung an die Peripherie und Anlage erfolgt. Es werden Daten der Anlage erfasst, sowie Echtzeitsteuerungen durchgeführt. Dies wird zumeist durch Speicherprogrammierbare Steuerungen (SPS) durchgeführt.

Die Aufgabe der Level-2-Automation ist es, die Funktion der Level-1-Automation zu optimieren, sowie Stellgrößen und Sollwerte auszugeben.

Die Datenerfassung beginnt gewöhnlich mit dem Level 1 und enthält die Koppelung an Messgeräte und Statusinformationen, wie Schalterstellungen die von dem SCADA-System erfasst werden. Die Daten werden dann in einer benutzerfreundlichen Darstellung präsentiert und ermöglichen es steuernd in den Prozess einzugreifen.

In diesem Projekt wird das Level 1 von einem Raspberry Pi (Raspberry Pi, 2014) und einem Arduino Mikrokontroller (Arduino, 2014) abgebildet. Diese dienen der Erfassung von Messwerten (Stromerzeugung, Stromverbrauch, Temperaturfühler) und dem Senden von Befehlen (Schalten der Heizung).

SIMATIC WinCC Open Architecture (SIMATIC WinCC Open-Architecture, 2014) ist ein SCADA-System für das Visualisieren und Bedienen von Prozessen, Fertigungsabläufen, Maschinen und Anlagen in allen Branchen. WinCC Open Architecture ist plattformneutral und verfügbar für Windows, Linux und Solaris.

Die im folgendem beschriebenen Merkmale haben dazu geführt WinCC Open Architecture als zentrales SCADA-System zu verwenden.

Durch die Client-Server-Architektur ist eine nahezu uneingeschränkte Skalierung des Systems gegeben. WinCC Open Architecture wird vom kleinen Einplatzsystem, bis zum verteilten und redundanten Mehrplatzsystem, in den unterschiedlichsten Konfigurationen eingesetzt.

- Multiuser- und Multitaskingsystem.
- Modularer Aufbau aus funktionsbezogenen Einheiten, den Managern, welche über TCP/IP angebunden werden.
- Die Kommunikation zwischen den einzelnen Prozessen erfolgt auf Basis von TCP/IP.
- Funktionalität und Last sind auf mehrere Rechner verteilbar.
- Das Mischen der Betriebssystemplattform (z. B. Linux und Windows) innerhalb eines Systems ist möglich.
- Ereignisorientierte und telegrammbasierende Kommunikation.
- Abgesetzte Clientarbeitsplätze werden über TCP/IP angebunden.

In WinCC Open Architecture gibt es für alle wesentlichen Funktionalitäten eigene autonome Ausführungseinheiten, die Manager. Unter einem Manager versteht man einen Prozess, welcher spezielle Aufgaben erfüllt. So gibt es beispielsweise eigene

Manager für die Peripherieankopplung, für die historische Datenhaltung oder für die Bedienoberflächen (User Interfaces).

- Event-Manager (EV) - bildet den Kern eines WinCC OA Systems.
- Treiber-Manager (D) - die Prozessanbindung erfolgt über Treiber.
- Data- Manager (DB) - speichert Prozessänderungen in einer Datenbank.
- Control-Manager (CTRL) - ist eine eigene Laufzeitumgebung, welche Programme einer stark vereinfachten, C ähnlichen, Programmiersprache eventgesteuert und multitaskingfähig abarbeitet.
- User Interface-Manger (UI) - sorgt für die grafische Visualisierung von Prozesszuständen.

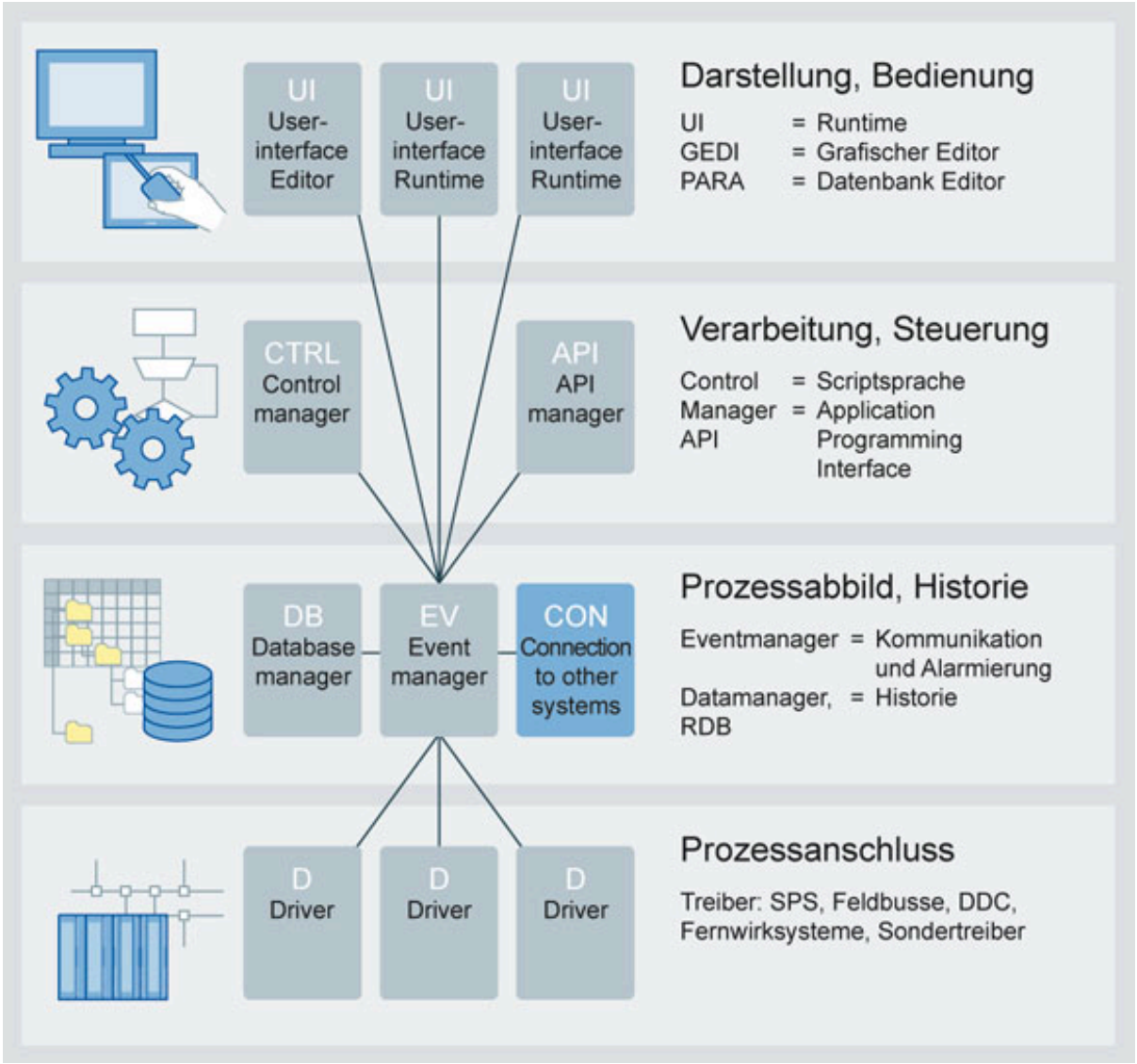



Abb. 4) Managerkonzept (SIMATIC WinCC Open Architecture Managerkonzept, 2014)

WinCC Open Architecture ist auf dem Raspberry Pi lauffähig, womit sich das Überführen der Level 1 Daten und Befehle realisieren lässt. Auf dem Raspberry Pi können abgesetzte WinCC Open Architecture Clientprogramme (Controlmanager) laufen, welche Daten vom lokalen System aufgreifen und an das zentrale System übermitteln. Befehle werden vom zentralen System an die lokalen Systeme weitergereicht. WinCC Open Architecture Controlmanagerprogramme können in einer stark vereinfachten C ähnliche Scriptsprache programmiert werden. Der Zugriff auf relationale Datenbanken aus der Scriptsprache ist, wie auch die automatisierte Datenarchivierung, in eine Oracle Datenbank möglich.

### 2.1.2 Oracle Datenbank als Datenspeicher

Oracle bietet mit Oracle XE (Oracle Database Express Edition, 2014) eine freie Version der Oracle Datenbank. Diese freie Version hat einen eingeschränkten Funktionsumfang und ist auf elf Gigabyte Benutzerdaten eingeschränkt. Für die Erfassung und Visualisierung aktueller Stromerzeugungsdaten ist der Umfang dieser Version ausreichend.

Oracle Application Express (Oracle Application Express, 2014) ist ein Werkzeug, mit welchem einfach und effizient Webanwendungen erstellet werden können und bietet integrierte Funktionalitäten zur Visualisierung von Zeitreihen. Bereits in der kostenlos erhältlichen Datenbankversion Oracle XE ist diese Entwicklungsplattform enthalten und benötigt zur Erstellung einer ansprechenden, schnellen und sicheren Applikation nur einen Browser.

Diese Webentwicklungsumgebung und die gute Integration mit dem SCADA-System WinCC Open Architecture, haben dazu geführt dass Oracle als Datenspeicherung gewählt wurde.

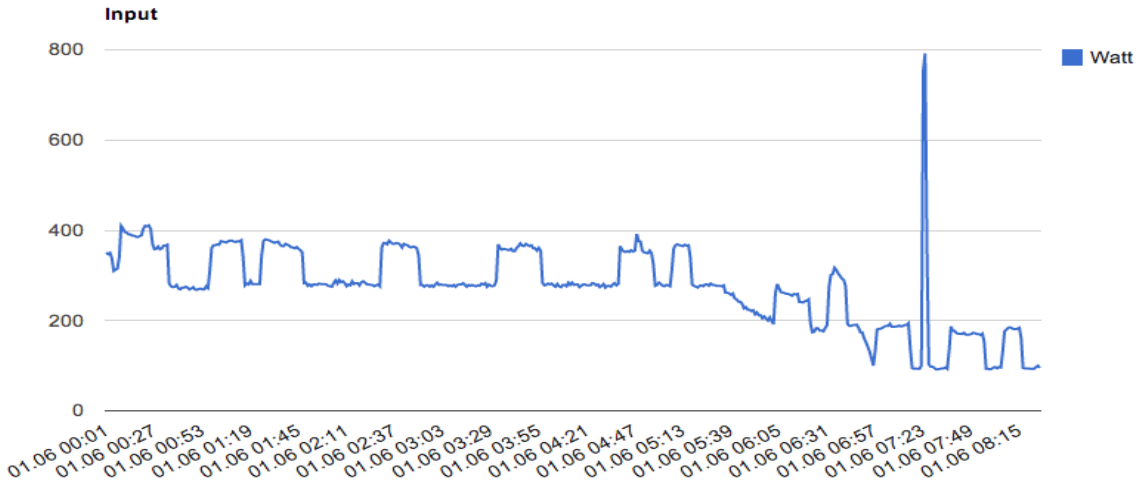



Abb. 5) Beispiel einer Zeitreihendarstellung mit Oracle Application Express

## 2.2 Erfassung der Stromerzeugungsdaten

### 2.2.1 Beschreibung des Wechselrichters

Als Wechselrichter wird ein Gerät bezeichnet welches die Gleichspannung aus Photovoltaikmodulen in Wechselspannung umwandelt und in das Haushaltstromnetz einspeist. Der Wechselrichter ist damit Teil einer Photovoltaikanlage. Auf der Eingangsseite befindet sich üblicherweise ein Gleichspannungswandler. Auf der Ausgangsseite befindet sich ein Wechselrichter, welcher in das Niederspannungsnetz einspeist und sich automatisch mit dem Stromnetz synchronisiert. Als Forschungsobjekt dient ein Wechselrichter Sunny Boy 4000tl der Firma SMA.


Abb. 6) SMA Wechselrichter

### 2.2.2 Datenerfassung über Bluetooth-Schnittstelle

Der Wechselrichter Sunny Boy 4000tl der Firma SMA besitzt eine Bluetooth-Schnittstelle zum Auslesen von Daten. Mit dem vom Hersteller frei verfügbarem Programm „Sunny Explorer“ können die Daten der Anlage per Bluetooth ausgelesen werden. Das Programm ist jedoch nur für Microsoft Windows erhältlich und bietet keine Schnittstellen um die Daten in ein Datenverarbeitungssystem zu übernehmen. Damit ist eine Verwendung und Integration in ein Gesamtsystem nicht möglich.

Das Bluetooth-Protokoll ist vom Hersteller nicht öffentlich zugänglich. Es gibt jedoch Open Source-Programme, welche das Protokoll implementiert haben und die Daten der Anlage auslesen können. Der Quellcode dieser Programme ist, unter den Bedingungen der Open Source-Lizenz, frei verfügbar und kann daher für eine Anbindung an ein Steuerungssystem verwendet werden (SMA Spot, 2014 & SMA Bluetooth, 2014).

Es wurde der Source Code von SMA Spot (SMA Spot, 2014) als Basis herangezogen, um die aktuellen Daten aus dem Konverter auszulesen. Der Sourcecode wurde auf einem Raspberry Pi übersetzt und in Betrieb genommen. Für die Installation müssen die benötigten und abhängigen Pakete installiert werden, sowie der Bluetooth-Stack „bluez“.

Zur Konfiguration muss die Bluetooth Adresse und das Passwort des Wechselrichters in das Konfigurationsfile eingetragen werden. Die Bluetooth Adresse ist unter Linux mittels „hcitool scan“ herauszufinden.

Das Programm liefert folgende CSV-Dateien mit Daten:

- „Spot-Daten“ ➔ Aktuelle Daten der Anlage zum Abfragezeitpunkt. Dabei bekommt man die Detailwerte der Anlage.

Tabelle 1: Spot-Daten der ausgelesenen CSV-Datei

| *Parameter* | *Beschreibung* | *Beispielwerte* |
|---|---|---|
| TS | Zeitstempel des Datensatzes | 07.06.2014 10:45:56 |
| PDC1_WATT | Gleichstrom Watt Leitung 1 | 1342 |
| PDC2_WATT | Gleichstrom Watt Leitung 2 | 1236 |
| IDC1_AMP | Gleichstrom Ampere Leitung 1 | 3.538 |
| IDC2_AMP | Gleichstrom Ampere Leitung 2 | 3.412 |
| UDC1_VOLT | Gleichstrom Volt Leitung 1 | 379.41 |
| UDC2_VOLT | Gleichstrom Volt Leitung 2 | 362.55 |
| PAC1_WATT | Wechselstrom Watt 1 | 2424 |
| PAC2_WATT | Wechselstrom Watt 2 | 0 |
| IAC1_AMP | Wechselstrom Ampere 1 | 10.321 |
| IAC2_AMP | Wechselstrom Ampere 2 | 0 |
| UAC1_VOLT | Wechselstrom Volt Strang 1 | 237.43 |
| UAC2_VOLT | Wechselstrom Volt Strang 2 | 0 |
| PDCTOT_WATT | Gleichstrom Gesamt (Eingang) | 2578 |
| PACTOT_WATT | Wechselstrom Gesamt (Ausgang) | 2424 |
| EFFICIENCY_PRC | Effizienz der Stromkonvertierung | 94.026 |
| ETODAY_KWH | Erzeugte kWh Tag | 2.978 |
| ETOTAL_KWH | Erzeugte kWh Gesamt | 6090.881 |
| FREQUENCY_HZ | Stromfrequenz (EU 50Hz) | 49.99 |
| OPERATINGTIME_HOUR | Anlagenlaufzeit | 5213 |
| FEEDINTIME_HOURS | Stromlieferungsstunden | 5096 |
| BT_SIGNAL_PRC | Bluetooth Signalstärke | 62.745 |

- Fünfminutenwerte ➔ Alle fünf Minuten werden akkumulierten Kilowattstunden (KWh) des Tages und die Leistung (Watt) geliefert.
- Tageswerte ➔ Die akkumulierten Kilowattstunden (KWh) des Tages.

Es wurden zwei Linux CRON–Jobs zur Erfassung der Daten angelegt. Ausgabe von „crontab –l“ auf dem Erfassungssystem:

```
* 6-22 * * * /data/sma/get_mins     #jede Minute die 5Min- und Spot-Werte.
0 5 * * * /data/sma/get_days       #jeden Tag um 5 Uhr die Vortageswerte.
```

Das Programm speichert die Werte in CSV-Dateien und hängt die gelesenen Werte an das Ende einer Datei. Mittels Parameter kann dem Programm mitgeteilt werden welche Daten geliefert werden sollen.

Shell-Script „get_mins“ zum Erfassen der Minutenwerte:

```
echo "-----------------------------------------------------" >> $0.out
echo "1" > /tmp/$0
echo `date`>> $0.out
/usr/local/bin/SMAspot -finq -ad1 -am0 >> $0.out 2>&1
echo "0" > /tmp/$0
echo `date`>> $0.out
```

In diesem Fall werden die Minutenwerte des aktuellen Tages (-ad1) abgefragt, zusätzlich liefert das Programm auch die Spotwerte und hängt die zum Abfragezeitpunkt aktuellen Werte an das Ende der CSV Datei.

Der Parameter „-am0“ bedeutet, dass keine Monatswerte abgefragt werden. Die Monatswerte werden nicht benötigt, da diese über die Tageswerte im übergeordneten Steuerungs-PC berechnet werden.

Der Parameter „-finq“ bedeutet, dass die Werte auf jeden Fall geliefert werden sollen, sonst liefert das Programm nur Werte während der Sonnenscheinstunden. Dies geschieht über Zeit und der Geolokation, welche vom Programm angenommen wird (Sonnenaufgang, Sonnenuntergang). Dieser Automatismus ist nicht notwendig, da die Erfassung zeitgesteuert aufgerufen wird.

### 2.2.3 Datenübertragung in die relationale Datenbank

Die von der Datenerfassung gelieferten Daten sind als CSV Textdateien vorhanden. Damit die Daten sinnvoll verarbeitet werden können und daraus Informationen abgeleitet werden können, werden die Daten in eine Oracle Datenbank übertragen. Dabei werden zuerst die Files an den Datenbankserver übertragen. Damit die Daten aktuell in der Datenbank zur Verfügung stehen, wird die Übertragung direkt nachdem die Files erzeugt wurden, angestoßen. Die Übertragung erfolgt in den bestehenden Shell-Scripts, und wird über SCP (Secure Copy) mit Private/Public-Key Authentifizierung durchgeführt.

#### 2.2.3.1 Einbinden der Dateien als externe Tabellen

Oracle bietet die Möglichkeit CSV-Dateien als externe Tabellen zu verwenden. Dabei wird eine Tabelle erzeugt welche auf eine CSV-Datei zeigt. Die Daten werden bei jeder Abfrage aus der CSV-Datei gelesen.

```
create table SMA_EXT_5MIN
(
  ts  VARCHAR2(30),
  kwh VARCHAR2(30),
  kw  VARCHAR2(30)
)
organization external
(
  type ORACLE_LOADER
  default directory SMA
  access parameters
  (
    RECORDS DELIMITED BY NEWLINE FIELDS TERMINATED BY ";"
  )
  location (SMA:'5min.csv')
)
reject limit UNLIMITED;
```

Die Daten werden im ersten Schritt als reine Textwerte zur Verfügung gestellt. In einem zweiten Schritt wird eine View erstellt, welche die Werte in das richtige Datenformat umwandelt.

```
create or replace view sma_act_5min as
select
  to_date(ts, 'DD/MM/YYYY HH24:MI:SS') ts,
  to_number(kwh, '9999999.999') kwh,
  to_number(kw, '9999999.999') kw
from sma_ext_5min;
```

Somit ist ein einfacher und direkter Zugriff auf die Daten aus der relationalen Datenbank möglich. Alle Abfragemöglichkeiten stehen mit dieser Tabelle zur Verfügung. Auch die analytischen Abfragemöglichkeiten von Oracle.

Die folgende Abfrage liefert zum Beispiel eine auflaufende Summe der Kilowattstunden über einen Abfragezeitraum:

```
select to_char(ts,'DD.MM HH24:MI') ts, sum(kw*1000/12) over (
  partition by trunc(ts) order by ts rows between
  unbounded preceding and current row
)  "KWh"
from sma_all_5min
where ts >= to_date(:P5_DATE1,'DD.MM.YYYY HH24')
and ts <= to_date(:P5_DATE2,'DD.MM.YYYY HH24')
order by ts
```

### 2.2.3.2 Datenimport der externen Dateien

Nachteil der externen Tabellen ist, dass auf diese Tabellen keine Indizes gelegt werden können. Damit werden Abfragen, nach Anhäufung von vielen Daten über einen längeren Zeitraum, langsam.

Daher werden die Daten aus den externen Tabellen einmal täglich in eine permanente (nicht externe) Oracle-Tabelle importiert. Dies erfolgt über einen Datenbankjob und PL/SQL Prozeduren.

Beispiel für den Import der Minutenwerte aus der externen Tabelle SMA_ACT_5MIN in die permanente Tabelle SMA_TAB_5MIN.

```
procedure import_mins
is
begin
  merge into sma_tab_5min d
  using (select ts, kwh, kw from sma_act_5min) s
  on (d.ts = s.ts)
  when not matched then insert (ts, kwh, kw) values (s.ts, s.kwh, s.kw);
end;
```

Nachdem das Laden der Daten erfolgt ist, können die CSV-Dateien gelöscht werden und für einen neuen Tag neu beschrieben werden. Damit wird sichergestellt, dass der Umfang der Daten in den CSV-Dateien auf einem gleichen Niveau bleibt.

Damit beim Zugriff auf die Daten über SQL nicht darauf achtgegeben werden muss in welcher Tabelle die benötigten Daten stehen (externe Daten aus den Dateien oder bereits importierte Daten), wird eine View über die externe und die permanente Tabelle erstellt, welche die Inhalte der beiden Tabellen miteinander vereint (Vereinigungsmenge).

### 2.2.4 Visualisierung der Stromerzeugungsdaten

Die Visualisierung der erfassten Stromerzeugungsdaten erfolgt über Oracle APEX. Wie im vorherigen Kapitel beschrieben werden die Photovoltaikdaten in die Oracle Datenbank importiert. Damit ist die Visualisierung über Oracle APEX aus der Oracle Datenbank direkt möglich.

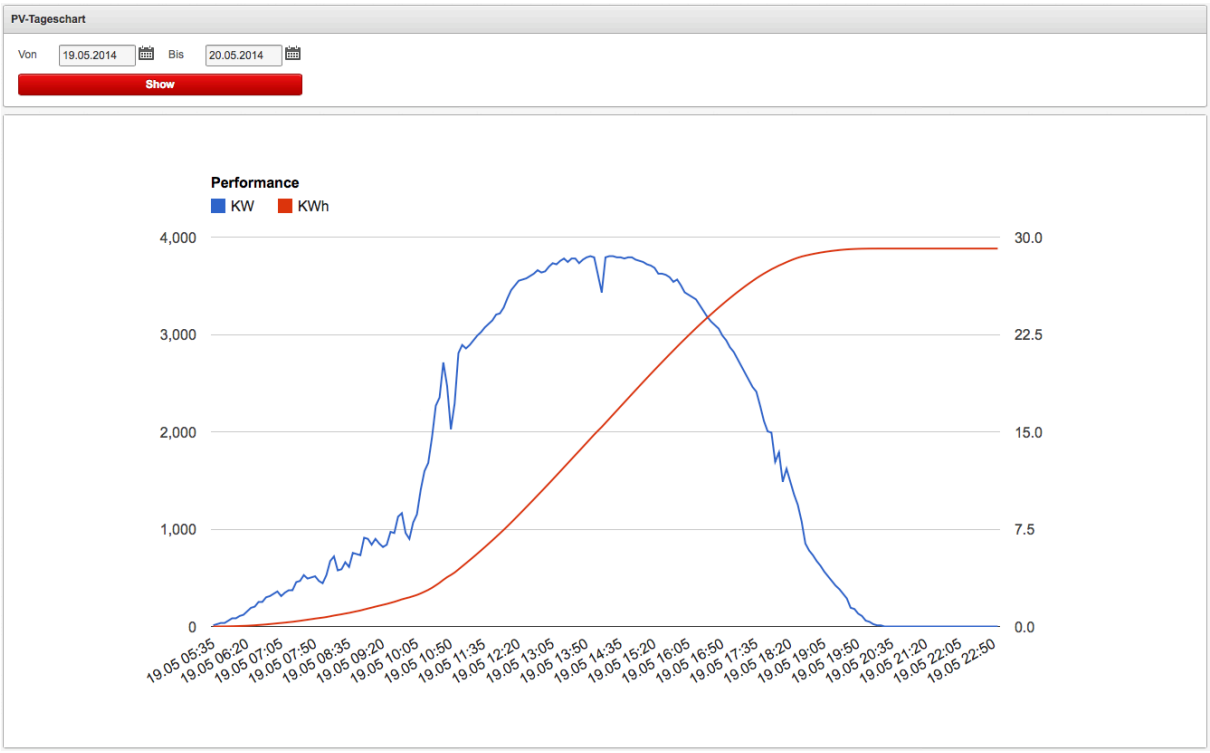



Abb. 7) Visualisierung der Stromerzeugung mit Oracle Applicaton Express

Die Visualisierung wird durch die folgende SQL-Abfrage realisiert:

```
SELECT a.ts AS TIMESTAMP, a.v AS Watt b.v AS KWh
  FROM (SELECT a.ts, NVL(a.KW, 0) * 1000 v
          FROM sma_all_5min a
         WHERE a.ts >= to_date(P_DATE1, 'DD.MM.YYYY')
           AND a.ts <= to_date(P_DATE2, 'DD.MM.YYYY') + 1) a,
       (SELECT a.ts,
               SUM(a.kw / 12) over(PARTITION BY trunc(a.ts)
                                   ORDER BY a.ts rows
                                   BETWEEN unbounded preceding AND
CURRENT ROW) v
          FROM sma_all_5min a
         WHERE a.ts >= to_date(P_DATE1, 'DD.MM.YYYY')
           AND a.ts <= to_date(P_DATE2, 'DD.MM.YYYY')) b
 WHERE a.ts = b.ts
   AND (a.v > 0 OR b.v > 0)
```

## 2.3 Erfassung der Stromverbrauchsdaten

Aufgrund der Installation einer Photovoltaikanlage werden vom Stromlieferanten zwei getrennte Zähler installiert. Einer zur Erfassung des Strombezuges vom Lieferanten und einer zur Erfassung der Stromlieferung an den Lieferanten. Diese Zähler haben jedoch keine digitale Schnittstelle, um Daten auslesen zu können.

Daher wurden zusätzlich drei Phasen Drehstromzähler der Marke Eltako vom Typ DSZ12DE mit einer S0-Schnittstelle verbaut. Die S0-Schnittstelle (DIN EN 62053-31, 2014) ist ein einfaches Impulsprotokoll, welches über einen Raspberry Pi erfasst und ausgewertet wird.

Es gibt Zwei-Wege-Drei-Phasen Drehstromzähler, welche den Stromverbrauch in beide Richtungen (Bezug/Lieferung) erfassen können. Jedoch bietet die S0-Schnittstelle nur ein PWM (Pulse Width Modulation) Signal (Wikipedia Pulsweitenmodulation, 2014), womit die Richtung, in welcher der Strom fließt, nicht erfassbar ist. Und somit nicht unterschieden werden kann ob die Leistung bezogen oder geliefert wird.

Daher wurden zwei Drehstromzähler verwendet. Einer zählt den Verbrauch und einer die Lieferung. Der Einbau der Drehstromzähler erfolgt dabei so, dass diese gegensätzlich gepolt verkabelt werden (+/-, -/+). Damit misst einer den Energiefluss in die eine Richtung und der zweite Zähler misst den Energiefluss in die andere Richtung.

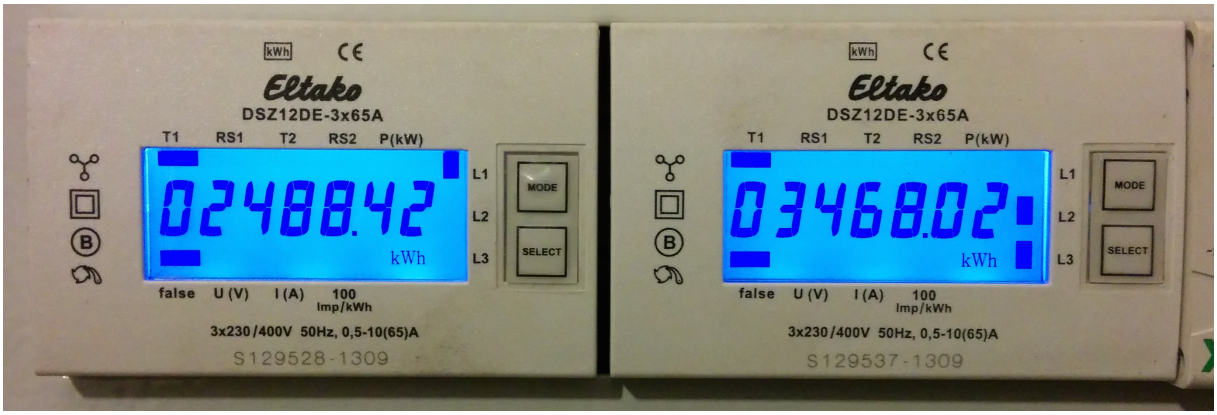



Abb. 8) Eltako DSZ12DE Drehstromzähler zum Erfassen von Strombezug und Stromlieferung

### 2.3.1 Datenerfassung mit dem Raspberry Pi

Die verwendeten drei Phasen Drehstromzähler sind mit einer S0-Schnittstelle ausgestattet. Die S0-Schnittstelle (DIN EN 62053-31, 2014) liefert ein PWM (Pulse Width Modulation) Signal (Wikipedia Pulsweitenmodulation, 2014), welches ein digitales Signal je verbrauchter Stromeinheit liefert. Die Leitung, welche dieses Signal liefert, wird mit einem digitalen Eingang des Raspberry Pi verbunden.

Das Impulssignal ist vom Drehstromzählerhersteller Eltako wie folgt spezifiziert:

- Pulse interface S0 according to DIN EN 62053-31
- Potential free by opto-coupler
- Max. 30V DC/20mA and min. 5V DC, impedance 100ohms,
- pulse length 30ms, 1000 Imp./kWh

Wesentlich wichtige Information zur Datenerfassung und Auswertung ist, dass 1000 Impulse je kWh geliefert werden (ein Impuls je Wattstunde). Damit kann über die Zeitdifferenz der Impulse die Leistung (Watt) errechnet werden ➔ 3600 / (Zeitdifferenz in Sekunden zum letzten Impuls). Bsp.: Wird das Impulssignal im Abstand von drei Sekunden geliefert ergibt das eine Leistung von 3600/3 = 1200 Watt (Leistung ist Arbeit/Energieumsatz pro Zeiteinheit).

Zur Erfassung der Signale wurde ein Java Programm entwickelt welches auf die digitalen Eingänge des Raspberry Pi horcht, einen aktuellen Leistungswert ermittelt und an das übergeordnete Steuerungssystem liefert.

Java wurde gewählt da für den Raspberry Pi eine umfangreiche Java Bibliothek zur Programmierung der digitalen Ein- und Ausgänge zur Verfügung steht (siehe 2.3.2 Programm zur Signalerfassung mit dem Raspberry Pi).

Für Auswertungen und Berechnungen sind Werte in definierten Zeitabständen besser zu verarbeiten und zu korrelieren. Daher wurde das Programm so implementiert, dass es zu definierten Zeitabständen aktuelle Werte liefert. Da die Stromzählimpulse zu nicht definierten Zeitabständen (da über die Länge der Zeitabstände der Wert definiert ist) liefern, mussten dazu die Werte in den definierten Zeitabständen summiert und aliquot abgerechnet werden. Ein Extremfall dabei ist, wenn die Leistung gegen Null geht. Der Stromzähler liefert dann nur mehr Signale in großen Zeitabständen (60 Watt = 1 Signal je Minute).

### 2.3.2 Programm zur Signalerfassung mit dem Raspberry Pi

Für die Umsetzung in Java wurde das Projekt Pi4J (The Pi4J Project, 2014) verwendet. Dieses bietet eine Bibliothek mit welcher digitale Eingänge des Raspberry Pi gelesen und gesetzt werden können. Auf digitale Eingänge kann ereignisgesteuert reagiert werden.

Beispiel eines ereignisgesteuerten Aufrufes einer Funktion, wenn ein Zählersignal anliegt:

```
GpioController gpio = GpioFactory.getInstance();

// provision gpio pins as an input pin with its internal pull down
resistor enabled
GpioPinDigitalInput pinInput
= gpio.provisionDigitalInputPin(RaspiPin.GPIO_03,
PinPullResistance.PULL_DOWN);

// create and register gpio pin listener
System.out.println("Listen ..." + meterIn);
pinInput.addListener(new GpioPinListenerDigital() {
    @Override public void
handleGpioPinDigitalStateChangeEvent(GpioPinDigitalStateChangeEvent
event) {
                if ( event.getState().isHigh() ) {
                    meterIn.signal(new Date());
                }
            }
        });
```

In diesem Beispiel wird die folgende Funktion zur Berechnung der Leistung aufgerufen:

```
public synchronized void signal(Date d) {
     currSigs++;
     if ( prevSignal.getTime() > 0 && prevPeriod.getTime() > 0 &&
currSigs == 1 ) {
         double diff = d.getTime() - prevSignal.getTime();
         double add = 1.0 / diff * (currPeriod.getTime() -
prevSignal.getTime());
         prevSigs += add;
         currSigs = (1-add);
         Debug.println("C " + meterId + " " + String.format("%3.3f
%3.3f", diff, add));
         dataHdl.addRecord(new DataRecord(meterId, new
Date(prevPeriod.getTime()), prevSigs));
         prevSigs=0;
     }
     prevSignal.setTime(d.getTime());
}
```

In dieser Funktion wird, über die Zeitdifferenz zum letzten Signal, die aktuelle Leistung ermittelt und über einen Datahandler in die Datenbank geschrieben.

Mit dem Aufruf der Funktion

```
dataHdl.addRecord(new DataRecord(meterId, new
Date(prevPeriod.getTime()), prevSigs));
```

wird das Ergebnis an das übergeordnete System übertragen.

### 2.3.3 Datenübertragung an das übergeordnete Steuerungssystem

Für die Datenübertragung an das übergeordnete Steuerungssystem wurde eine REST (Representational State Transfer) Web API Schnittstelle gewählt.

Das Java Programm kann damit die aktuellen Stromverbrauchswerte an das SCADA-System sowie auch an die Oracle Datenbank übertragen. Beide bieten RESTful Services zur Datenerfassung.

Als Übertragungsinhalt wurde das Format JSON gewählt. Dieses kann sowohl im SCADA-System, als auch in der Oracle Datenbank, aufgrund von bestehenden Bibliotheken gelesen und interpretiert werden.

Die Datenübertragung läuft im Programm als eigener Thread, damit die Datenübertragung den Erfassungsprozess im Programm nicht beeinflusst oder blockiert. Die Daten werden über den Funktionsaufruf „addRecord“ in eine lokale Wartenschlange (Queue) abgelegt. Der Verarbeitungsthread läuft ständig im Hintergrund und überwacht die Warteschlange auf neue Daten. Es wird nicht mit jedem Funktionsaufruf ein Thread gestartet, da das Erzeugen eines Threads ein relativ aufwändiger Prozess ist und am Raspberry Pi nur eine begrenzte Leistung zur Verfügug steht.

Es wurde zusätzlich ein lokaler Puffer implementiert. Damit kann der Raspberry Pi die Stromerzeugungsdatenerfassung auch bei einer fehlenden Verbindung zum übergeordneten System durchführen und lokal speichern. Ist die Verbindung zum übergeordneten System nicht vorhanden werden die Daten nicht verworfen, sondern in eine, am Raspberry Pi laufende, MySQL Datenbank zwischengespeichert. Wird bei der nächsten angeforderten Datenübertragung festgestellt, dass die Verbindung zum übergeordneten System wieder vorhanden ist, werden die in der MySQL Datenbank zwischengespeicherten Daten ausgelesen und an das übergeordnete System mitübertragen.

Damit kann die Stromzählerdatenerfassung autark und ohne Datenverlust bei einer fehlenden Verbindung zwischen den Systemen arbeiten. Der Raspberry Pi kann

damit die erfassten Stromzählerdaten auch an ein entferntes System (z.B. Cloud) oder an ein nicht permanent zur Verfügung stehendes System senden. Zum Beispiel über eine GPRS Datenverbindung (GPRS General Packet Radio Services, 2014). Damit kann der Raspberry Pi als intelligenter Stromzähler auch in geoverteilten Systemen (verteilte Solarstationen) zum Einsatz kommen.

## 2.4 Warmwasserboiler als Energiespeicher

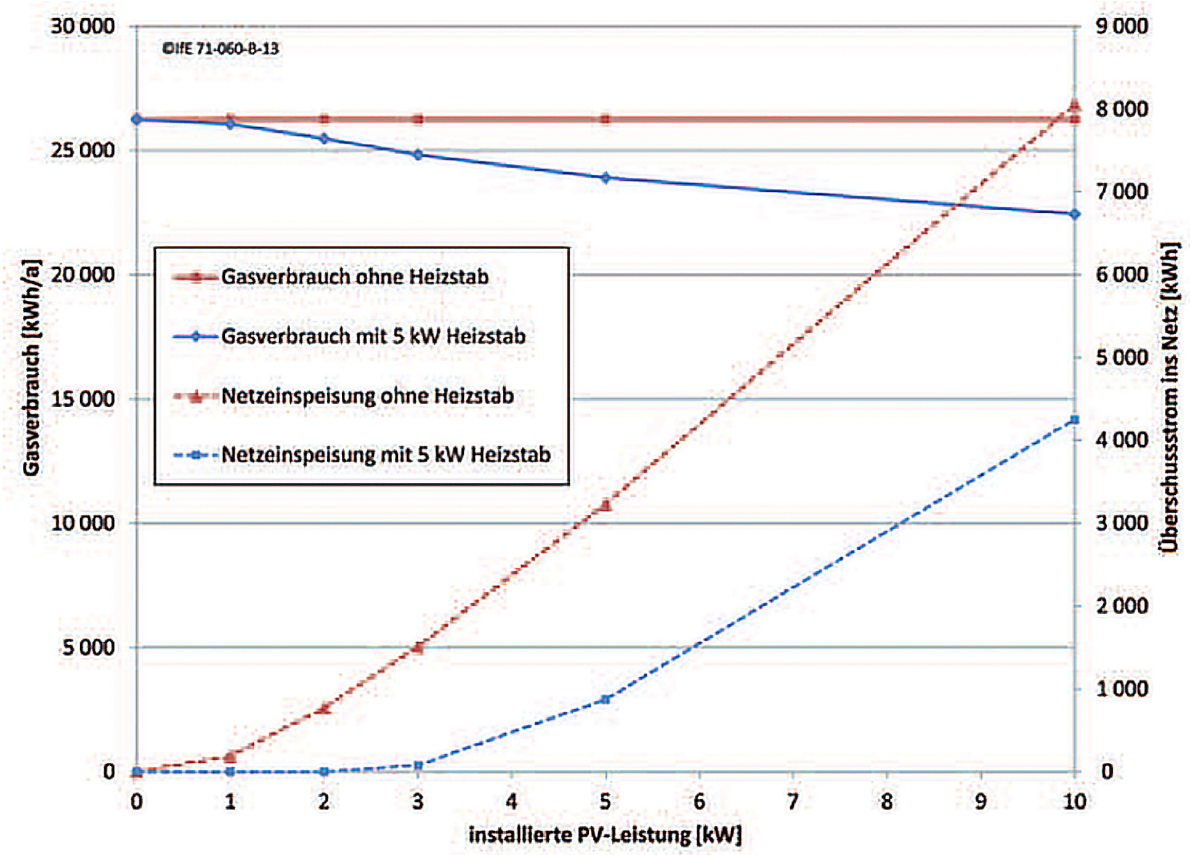



Abb. 9) Möglichkeiten zur Verwertung von Überschussstrom im Wärmesystem in Abhängigkeit der installierten PV (Photovoltaik)-Leistung. Die roten Linien stellen den Gasverbrauch und die Netzeinspeisung ohne Einsatz eines Heizstabs dar, die blauen Linien zeigen den Fall mit Einsatz eines Heizstabs (Huber et al., 2012, S. 59)

In Szenarien wurden zwei Alternativen zur Verwendung des Überschussstroms untersucht. Die Speicherung in Batterien sowie die Verwendung im Wärmesektor durch einen Heizstab. Diese Technologien wurden mittels der angewandten Optimierung kostenoptimal in den Szenarien eingesetzt. Ein wesentlicher Vorteil der Lösung im Wärmesektor durch einen Heizstab ist in den niedrigen Anschaffungskosten und der einfachen Integrierbarkeit in existierende Heizungssysteme zu sehen (Huber et al., 2012, S. 59-60).

Daher wird in diesem Projekt zur thermischen Speicherung der Warmwasserboilers eines Einfamilienhauses herangezogen. Der Warmwasserboiler wird im Normalbetrieb mit Gas betrieben. Über eine elektrische Zusatzheizung wird das Aufheizen mit selbst erzeugten Photovoltaikstrom erfolgen. Damit wird das Aufheizen über die Gastherme minimiert, wodurch der Gasverbrauch sinkt und die Kosten für Gas reduziert werden.

Ziel ist, dass das Warmwasser im Warmwasserboiler über eine elektrische Zusatzheizung aufgeheizt wird. Damit dies intelligent erfolgen kann, muss die Temperatur des Wassers im Boiler als Regelgröße gemessen und in das System übertragen werden. Damit kann das System, anhand der Temperatur, entscheiden ob ein Aufheizen notwendig ist.

Abb. 10) Warmwasserspeicher mit Heizstab

### 2.4.1 Regelgröße des Warmwasserboilers

Der Warmwasserboiler soll nur dann geheizt werden wenn das Warmwasser einen definierten Temperaturwert unterschreitet. Damit diese Regelgröße in das System geführt werden kann, muss die Temperatur des Warmwasserboilers mit einem Temperaturfühler erfasst werden. Zur bestehenden visuellen Temperaturanzeige wird im selben Schacht, welcher zur Mitte des Boilers führt, ein analoger Temperaturfühler eingeführt. Dieser analoge Temperaturfühler wird an einen Mikrokontroller angeschlossen, über welchem die Temperatur erfasst wird.

### 2.4.2 Messung der Gasthermenaktivität

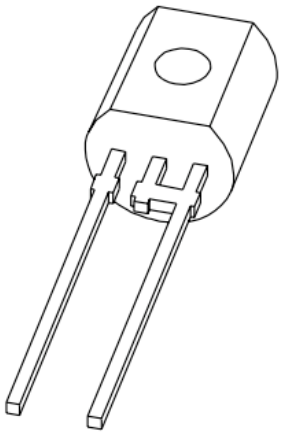
Der Gaszähler besitzt keine Möglichkeit zum digitalen Auslesen des Verbrauches. Die Erfassung ist jedoch zur Eruierung des verbrauchten bzw. weniger verbrauchten Gases von Interesse. Damit kann eruiert werden, welche Einsparung die automatisierte Heizungssteuerung mit einem elektrischen Heizstab bringt.

Als Alternative wird die Temperatur des Wassers gemessen, welches von der Gastherme in den Boiler fließt. Durch einen Temperaturanstieg des Wassers ist erkennbar, dass die Gastherme heizt. Damit kann aufgezeichnet werden, wann und wie lange die Gastherme das Warmwasser des Boilers heizt.

Dafür wird ein analoger Temperaturfühler an den Warmwasserausgang der Gastherme geheftet. Dieser analoge Temperaturfühler wird an einen Arduino Mikrokontroller (siehe Kapitel 2.5.1 Temperaturfühlererfassung mit dem Arduino) angeschlossen und über diesen die Temperatur erfasst.

## 2.5 Temperaturerfassung des Warmwasserboilers

Als Temperaturfühler wurden Philips KTY81-2 Temperatursensoren verwendet (Philips KTY81-2, 2014). Die Kosten eines Temperaturfühlers liegen unter zwei Euro. Der Temperatursensor bietet einen annähernd linearen Verlauf des Widerstandes über den Temperaturmessbereich und ist daher für die Aufnahme der Temperatur gut geeignet.



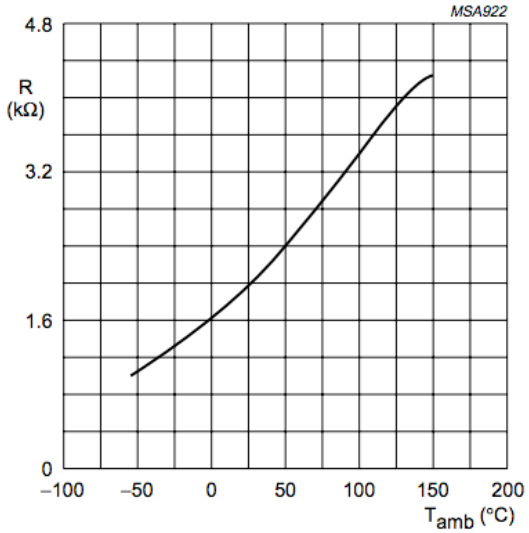



Abb. 11) Philips KTY81-2 Temperaturfühler mit Widerstandskurve

### 2.5.1 Temperaturfühlererfassung mit dem Arduino

Der Temperaturfühler KTY81-2 ist ein analoger Temperaturfühler, welcher durch seinen temperaturabhängigen Widerstand die angelegte Spannung verändert. Zur Datenerfassung wird ein Erfassungsgerät benötigt welches eine angelegte Spannung messen kann. Ein Raspberry Pi bietet mit seinem GPIO (General Purpose Input Output) die Möglichkeit digitale Eingänge zu erfassen, jedoch nicht die Möglichkeit analoge Eingänge zu messen. Es wurde daher ein Arduino Mikrokontroller für die Erfassung der Temperaturfühler gewählt. Dieser besitzt neben digitalen Eingängen auch analoge Eingänge.

Zum Ablesen eines analogen Einganges kann die Funktion „analogRead()“ verwendet werden. Diese Funktion ist Bestandteil der Programmierbibliothek des Arduino.

analogRead(): Liest den Wert des angegebenen analogen Pins. Das Arduino-Board enthält einen 6-Kanal (8 Kanäle beim Mini und Nano, 16 Kanäle beim Mega), 10-Bit Analog/Digitalwandler. Das bedeutet Eingangsspannungen zwischen 0V und 5V werden in ganzzahlige Werte zwischen 0 und 1023 umgesetzt. Das ergibt eine Auflösung von: 5V / 1023 Schritte oder 0,0049 Volt (4,9 mV) pro Schritt. Die Eingangsbandbreite und die Auflösung können über analogReference() verändert werden. Der Mikrokontroller benötigt circa 100 Mikrosekunden um einen anlogen Input zu lesen, das ergibt also maximal 10.000 Lesezyklen pro Sekunde (Arduino Funktion analogRead, 2014).

Damit der Temperaturfühler richtig ausgelesen werden kann, muss vor der Eingangsspannung ein Widerstand von 4.7KOhm geschaltet werden. Nach diesem Widerstand befindet sich die Verbindung zum analogen Eingang.

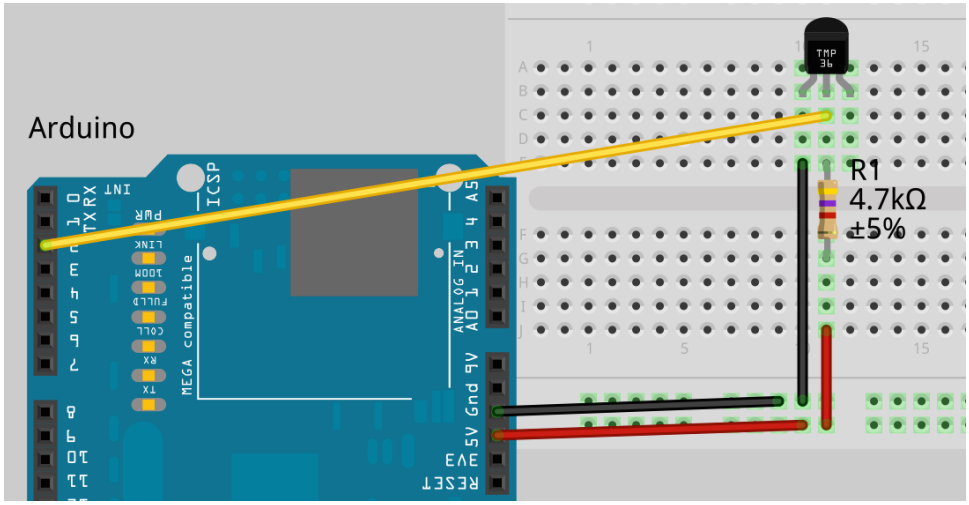



Abb. 12) Temperaturfühlerverdrahtung mit dem Arduino (Arduino, 2014)

### 2.5.2 Datenübertragung an das übergeordnete Steuerungssystem

Der Mikrokontroller Arduino hat einen USB-Anschluss, mit diesem kann ein Rechner verbunden werden. Der USB-Anschluss wird zur Programmierung und Programmübertragung verwendet, und kann von Programmen auf dem Arduino als emulierte serielle Schnittstelle verwendet werden. Über diese emulierte serielle Schnittstelle werden die erfassten Daten an einen Rechner übertragen.

Der Arduino dient in diesem Projekt als Kommunikationsschnittstelle zur realen Außenwelt, der Erfassung von Temperaturmesswerten. Es wurde ein Programm für den Arduino implementiert mit welchem die Schnittstellen konfiguriert werden können und welches die vom Arduino erfassten Messwerte über die serielle Schnittstelle überträgt.

Der Arduino wird über die emulierte serielle Schnittstelle mit dem bereits vorhandenen Raspberry Pi verbunden. Auf dem Raspberry Pi läuft ein Controlmanager des SCADA-Systems WinCC Open Architecture. Dieser Controlmanageer kommuniziert mit dem Arduino über die serielle Schnittstelle und ist selbst mit dem SCADA-System über TCP/IP verbunden.

In WinCC Open Architecture wurde ein Datenpunkttyp für analoge Eingänge angelegt. Über Datenpunkte von diesem Datenpunkttyp können die analogen Eingänge des Arduino konfiguriert werden. WinCC Open Architecture ist objektorientiert. Ein Datenpunkttyp entspricht einer Klasse, von welcher Instanzen (Datenpunkte) angelegt werden können.

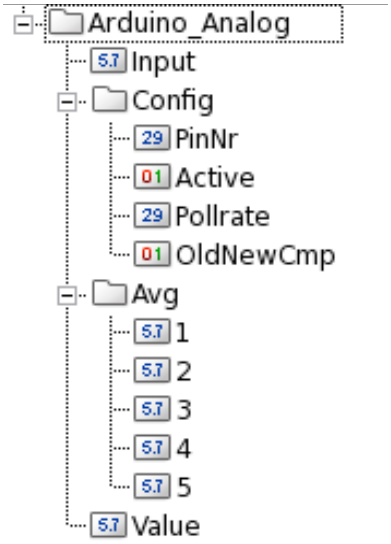



Abb. 13) Datenpunkttyp eines analogen Einganges

Folgende Konfigurationsmöglichkeiten für einen analogen Eingang bestehen:

- PinNr: Definiert den Pin am Arduino zu welchem der Datenpunkt verbunden wird.
- Active: Soll der Pin vom Programm gelesen und übertragen werden.
- Pollrate: Abtastrate des analogen Einganges in Millisekunden.
- OldNewCmp: Soll vom Erfassungsprogramm am Arduino ein Alt-/Neuvergleich durchgeführt werden. Wird ein Alt-/Neuvergleich durchgeführt, dann wird ein Wert vom Arduino nur dann über die serielle Schnittstelle übertragen wenn sich der Eingangswert ändert.

Wenn der Datenpunkt aktiv geschaltet ist, werden die Messwerte des konfigurierten Einganges vom Arduino über den Controlmanager an den Datenpunkt übertragen.

- Input: Eingangs-/Rohwert des analogen Einganges.
- Value: Ingenieurwert, welcher bei Bedarf im SCADA-System mittels einer Wertumrechnung vom Eingangs-/Rohwert errechnet wird.
- Avg.*: Vom SCADA-System berechnete Durschnittwerte (10 Sekundenwert, 30 Sekundenwert, Minutenwert, Stundenwert).

Für die Kommunikation zwischen dem Ardunio und dem SCADA-System wurde ein Programm in WinCC Open Architecture implementiert. Dieses Programm sit die Schnittstelle zwischen den beiden Komponenten. Einerseits werden die Konfigurationsdaten (PinNr, Active, Pollrate, OldNewCmp), mit jeder Änderung, zum Programm des Arduino übertragen, und andererseits werden die vom Arduino-Programm gelieferten Messwerte mit Zeitstempel auf die entsprechenden Datenpunktelemente in das SCADA-System übertragen.

Auszug aus dem Ardunio-Sourcecode für das Senden von Digitalwerten im definierten Pollzyklus, inklusive Alt/Neuvergleich und dem anschließendem Senden über die serielle Schnittstelle:

```
for ( i=0; i<14; i++ ) // Pins 1..14
{
  if ( digpin_active[i] &&
       (digpin_rate[i]==0 || ts_curr >= digpin_next[i] || ts_curr <
ts_last) )
  {
    b=digitalRead(i);
    if ( !digpin_oldnew[i] || b != digpin_value[i] ) // Alt/Neuvergleich
    {
      digpin_value[i]=b;
      digval_send(i, b, millis());
    }
    digpin_next[i]=ts_curr+digpin_rate[i];
  }
}



void digval_send(int i /*pin*/, bool b, unsigned long ms)
{
 unsigned long d=0;
 if ( b ) {
  if ( digpin_last_high[i] > 0 )
   d=(ms>digpin_last_high[i]) ? ms-digpin_last_high[i] : MAXULONG-
digpin_last_high[i]+ms;
  digpin_last_high[i]=ms;
 }
 else {
  if ( digpin_last_low[i] > 0 )
   d=(ms>digpin_last_low[i]) ? ms-digpin_last_low[i] : MAXULONG-
digpin_last_low[i]+ms;
  digpin_last_low[i]=ms;
 }
 Serial.write(STX); Serial.write('D');
 Serial.print(";"); Serial.print(i);
 Serial.print(";"); Serial.print(b ? "1" : "0");
 Serial.print(";"); Serial.print(ms);
 Serial.print(";"); Serial.print(d);
 Serial.write(ETX);
}
```

### 2.5.3 Umrechnung Spannungswert in Messwert

Im Falle des Temperaturfühlerwertes wurde folgende Umrechnung vom Spannungswert (x) in einen Messwert (y) und Temperaturwert (t) gewählt:

```
Messwert y = 1024-x        Temperatur t = 0,8673 * y - 560,31
```

Der Spannungswert eines analogen Einganges am Arduino ist ein Wert zwischen 0 und 1024. Dieser Wert ist die Abbildung der Eingangsspannung zwischen 0V und 5V. Bei dem Temperaturfühler handelt es sich um einen Widerstand, welcher bei steigender Temperatur seinen Widerstand erhöht. Dies bedeutet, dass sich der Wert umgekehrt proportional zur Temperatur verhält. Steigt die Temperatur, steigt der Widerstand, wodurch sich die Spannung vermindert.

Um einen Temperaturwert in Grad Celsius zu erhalten muss der Temperaturfühler kalibriert werden. Dazu wurden, über die am Warmwasserboiler vorhandene analoge Temperaturanzeige, Temperaturwerte und zugehörige Spannungswerte des analogen Einganges abgelesen.

Tabelle 2: Spannungswerte und abgelesene Temperaturwerte

| *Wert analoger Eingang* | *Temperatur* |
|---|---|
| 715 | 60° |
| 712 | 57° |
| 707 | 53° |

Aus den abgelesenen Werten kann über Interpolation (lineare Regression) eine Umrechnung von Eingangswert zu Temperaturwert errechnet werden:

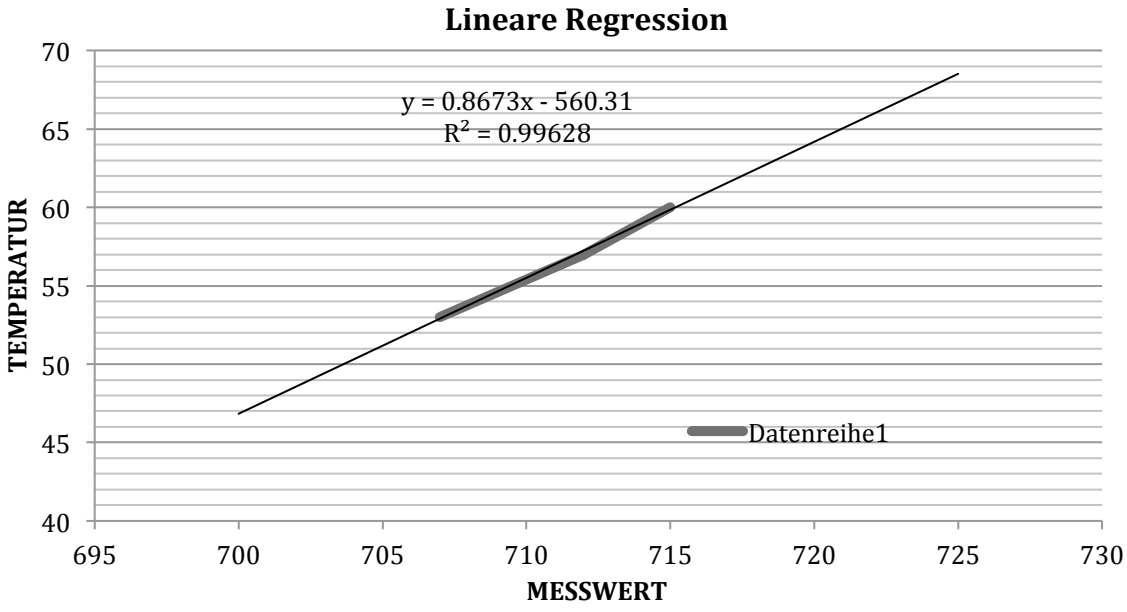



Abb. 14) Interpolation Messwert zu Temperaturwert

## 2.6 Endgerätesteuerung mit Funksteckdosen der Firma ELRO

Das Wasser des Warmwasserboilers wird über eine elektrische Zusatzheizung geheizt. Die Schaltung der elektrischen Zusatzheizung wird über Funksteckdosen der Firma ELRO durchgeführt. Varianten der Funksteckdosen sind mit einer Maximalleistung von bis zu 3000 Watt erhältlich. Damit kann auch der verwendete Heizstab mit einer Leistungsaufnahme von max. 2500 Watt gefahrlos geschaltet werden.


Abb. 15) ELRO Funksteckdose mit USB Dongle-Modul

Zur automatisierten Steuerung über einen PC wird ein erhältliches USB Dongle-Modul verwendet. Mit diesem können die Funksteckdosen über ein implementiertes Programm geschaltet werden.

Beschreibung des Herstellers:

Zum Steuern aller Empfänger der HOME EASY und AB600 Serien.

- Ideal zum Ein/Ausschalten der Schalter und Dimmer zu einem festgelegten Zeitpunkt über PC
- Inkl. Windows Software
- Frequenz: 433.92MHz
- Funkreichweite bis zu 50 Meter
- Signal durchdringt Fenster, Türen, Wände und Decken
- verbindbar mit allen Home Easy Unterputz- und steckfertigen Schaltern und Dimmern

Die vom Hersteller mitgelieferte Software läuft auf einem Windows-PC und ist einfach gehalten. Die Software bietet keine Schnittstellen zur Ansteuerung über Fremdsysteme.

Daher wurde zum Ansteuern des USB Dongle-Moduls ein C++ Programm entworfen. Dazu wurde das USB Protokoll eruiert und mittels einer Multiplattform USB Library (HIDAPI - Multi-Platform library for communication with HID devices) implementiert.

Das Programm ist auf dem Raspberry Pi lauffähig und kann mit Standardmethoden von Linux von einem entfernten Rechner aufgerufen werden (Remote Program Call über SSH).

Dieser Aufruf erfolgt vom SCADA-System WinCC Open Architecture aus einem Controlmanager, welcher auf einen Datenpunkt verbunden ist. Wird der Datenpunkt im SCADA-System mit einem Wert (Ein/Aus) verschickt, so reagiert der Controlmanager auf diese Wertänderung und ruft das Kommunikationsprogramm für den USB-Dongle mit entsprechenden Parametern auf und schaltet dadurch die Funksteckdose.

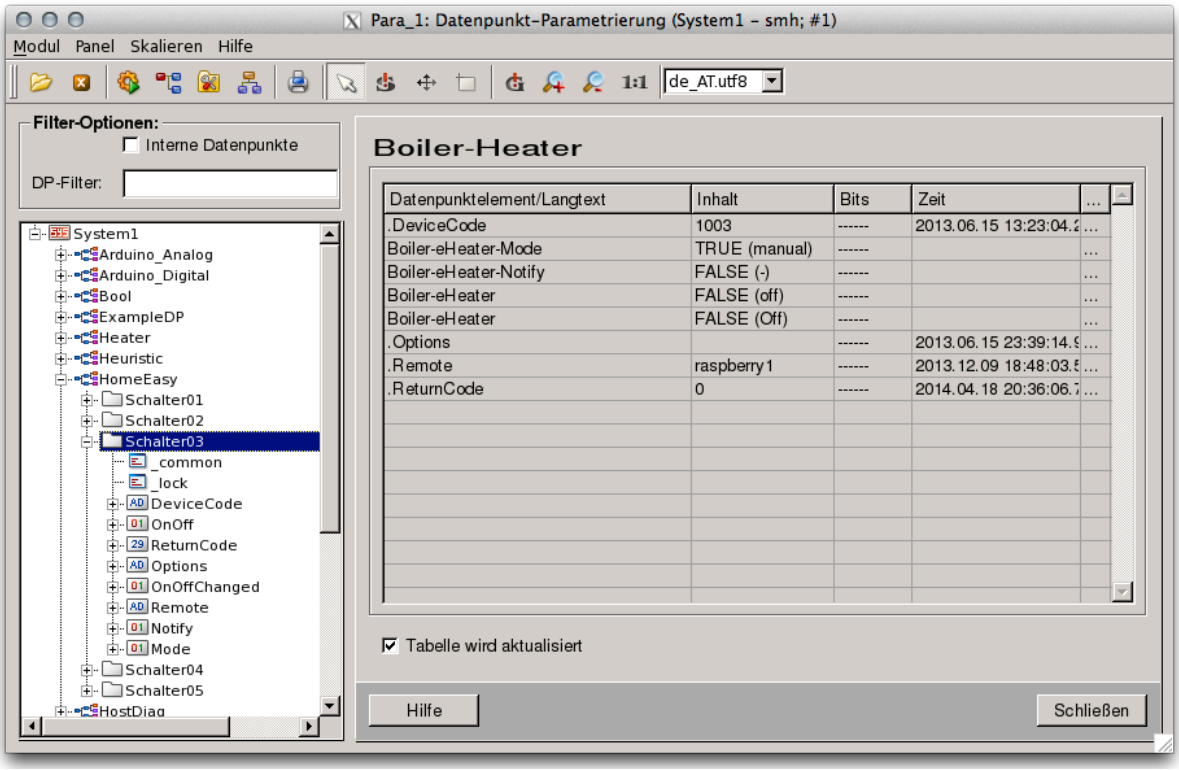



Abb. 16) Datenpunkt zum Steuern der ELRO Funksteckdose

# 3 Integration der Systemkomponenten

In den vorherigen Kapiteln wurden die Teilkomponenten und deren Umsetzung beschrieben. Für die gewünschte Steuerung der Heizung wird ein Steuerungssystem, welches die Regelaufgaben übernimmt, benötigt. Dieses führt die Teilkomponenten zu einem Gesamtsystem zusammen.

Das Steuerungs- und Regelsystem bildet somit den Kern des Systems. Es benötigt zur Erfüllung seiner Aufgabe die Stromerzeugungsdaten, die Stromverbrauchsdaten und die Daten des Energiespeichers. Aus diesen Informationen wird die Stellgröße (Schalten der Heizung) abgeleitet und die Aktion an das Peripheriesystem gesendet.

Die zentrale Steuerung erfolgt über das SCADA-System WinCC Open Architecture. Wie in den vorherigen Kapiteln beschrieben, werden alle benötigten Daten aus den verteilten Teilsystemen in das SCADA-System übertragen und gespeichert. Im SCADA-System wurde ein Controlmanager implementiert welcher sich auf die Eingangsgrößen verbindet und diese auswertet.

Damit die Regelung bei Unregelmäßigkeiten nicht unmittelbar schaltet, wird ein gleitender Durchschnitt des Energieflusses gebildet. Dieser bildet die Regelgröße für die Regelung. Der Energiefluss wird über die drei erfassten Stromwerte (Stromeingang, Stromausgang, Photovoltaik) berechnet.

Durch die Glättung wird vermieden dass kurze Einbrüche der Stromproduktion zur Abschaltung eines aktiven und trägen Heizprozesses führen. Im umgekehrten Fall führen kurze Stromproduktionsschübe nicht zum Einschalten des Heizprozesses. Beides kombiniert, unterdrückt ein häufiges Ein-/Ausschalten des Steuerungsprozesses.

## 3.1 Steuerung und Regelung der Heizungsanlage

Die Steuer- und Regelungskomponente hat die Aufgabe die Informationen aus der Datenverarbeitungskomponente zu lesen und in Abhängigkeit davon, die Heizung ein- und auszuschalten. Das Ein- und Ausschalten läuft über die Teilkomponente der Endgerätsteuerung.

Das Steuerprogramm wertet folgende Eingangsgrößen aus:

- Gleitender Fünfminutendurchschnitt des Energieflusses
- Minutendurchschnittswert der Warmwasserboiler-Temperatur

Für den Teil der Regelung wurde, wie in der Bachelorarbeit 1 „Eigenverbrauchsoptimierung von Photovoltaikstrom in Einfamilienhäusern." Vogler, A. (2014) beschrieben, ein entsprechender Algorithmus implementiert.

Der Algorithmus wurde ereignisgesteuert implementiert, das heißt dass das Programm bei jeder Änderung der Eingangsgrößen ausgeführt wird und eine Steuergröße berechnet.

Zu beachten ist, dass die elektrische Heizung die Gasheizung nicht komplett ersetzen kann. Die elektrische Heizung soll grundsätzlich nur betrieben werden, wenn Energie aus der Photovoltaikanlage bezogen werden kann. Ist nicht genug Energie aus Eigenproduktion für das Heizen vorhanden, dann soll das Warmwasser zumindest auf eine Temperatur von ca. 45° durch die Gastherme geheizt werden. Die Regelung der Gasheizung kann nicht direkt beeinflusst werden, da es sich um eine eigenständige Regelung ohne Anschlüsse zur Steuerung oder Überwachung handelt. Daher wurde die Gastherme manuell auf eine Mindestsolltemperatur eingestellt.

Die Gasheizung liegt im unteren Bereich, die elektrische Heizung im mittleren Bereich des Boilers. Dadurch kann es dazu kommen, dass trotz elektrischer Heizung die Gastherme, aufgrund der Solltemperaturunterschreitung, zu heizen beginnt. Dies ist dadurch zu erklären, dass warmes Wasser aufsteigt und sich die Wärme zuerst im oberen Bereich verbreitet und es dadurch zu einer Unterschreitung der Solltemperatur der Gastherme kommen kann. Die Häufigkeit ist jedoch gering, und wird mit dem Bedacht auf sicher vorhandenes Warmwasser im Falle einer ausbleibenden elektrischen Heizung, hingenommen.

## 3.2 Steuerprogramm für die Warmwasserheizung

Für die Steuerung und Konfiguration des Programmes wurde ein Datenpunkt mit Datenpunktelementen im SCADA-System angelegt. Über diesen kann das Steuerprogramm konfiguriert, beeinflusst und überwacht werden.

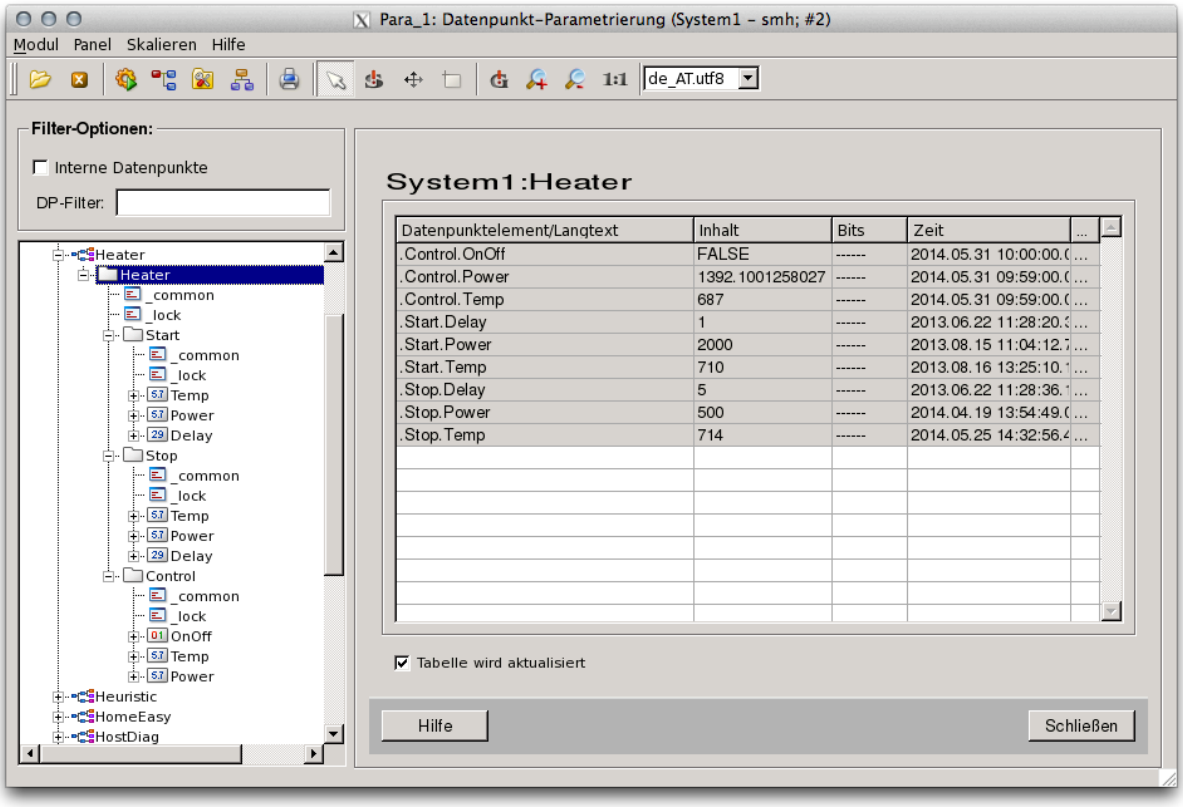



Abb. 17) Datenpunktstruktur der Heizungssteuerung

Der Datenpunkt gliedert sich in folgende Bereiche auf:

- Control ➔ Steuerung und Regelgrößen des Steuerprogrammes:
  - OnOff ➔ Datenpunkt zum Ein-/Ausschalten der Heizung. Dieser wird an das Programm zur Endgerätesteuerung weitergeleitet.
  - Temp ➔ Temperatur und Regelgröße für die Regelung. Dieser wird vom Temperaturerfassungsprogramm beliefert.
  - Power ➔ Zur Verfügung stehender Strom. Über die Werte der Programme der Stromerzeugungsdatenerfassung und der Stromverbrauchsdatenerfassung wird ein Energiefluss errechnet. Energiebezug wird als negativer Wert, Energielieferung(Energieüberschuss) wird als positiver Wert abgebildet.

- Start ➔ Konfiguration für das Starten der Heizung:
  - Temp ➔ Bei Unterschreitung dieses Temperatursensorwertes soll ein Aufheizen erfolgen.
  - Power ➔ Mindestenergieüberschuss, welcher vorhanden sein muss, um das Aufheizen zu starten.
  - Delay ➔ Der Zustand „Heizungsstart“ muss diese Zeitdauer anstehen bevor die Heizung gestartet wird. Dies dient zur Flatterunterdrückung.

- Stop ➔ Konfiguration für das Stoppen der Heizung:
  - Temp ➔ Bei Überschreitung dieses Temperatursensorwertes wird das Aufheizen abgeschaltet.
  - Power ➔ Abschaltung der Heizung bei Unterschreitung dieses Energieüberschusses.
  - Delay ➔ Der Zustand „Heizungstopp“ muss diese Zeitdauer anstehen bevor die Heizung gestoppt wird. Dies dient zur Flatterunterdrückung.

In Pseudocode vereinfacht dargestellter Algorithmus:

```
Wenn die Heizung nicht läuft
und [Control.Power] >= [Start.Temp] ist
und [Control.Temp] <= [Start.Temp] ist
➔ dann wird die Heizung eingeschaltet.

Wenn die Heizung läuft
und [Control.Power] <= [Stop.Power] ist
oder [Control.Temp] >= [Stop.Temp] ist
➔ dann wird die Heizung ausgeschaltet.
```

## 3.3 Verifizierung des Algorithmus zur Heizungsregelung

An folgendem Beispiel ist erkennbar wie die Regelung aufgrund der Regelungsgrößen, nach dem definierten Algorithmus, die elektrische Heizung ein- und ausschaltet.


Abb. 18) Verifizierung des Algorithmus anhand Messwertaufzeichnungen

In der obigen Darstellung ist die produzierte Energie als blaue Kurve (PV Watt 1min [Watt]) dargestellt, der Energiekonsum als braune Linie (Energy-Consumption [Watt]), sowie die Temperatur (Boiler-Sensorwert) des Warmwasserboilers. Die rote Stufenlinie (Boiler-eHeater) kennzeichnet den Status der Heizung (Ein/Aus).

Es ist erkennbar dass ab der Überschreitung der eingestellten Sollüberschussenergie die Heizung um 11:05 eingeschaltet wurde. Nachdem die Heizung eingeschaltet wurde, ist ein deutlicher Anstieg des Energiekonsums (orange Linie) erkennbar (verursacht durch die elektrische Heizung) sowie auch der Anstieg der Temperatur im Warmwasserboiler.

Die darauffolgende automatische Abschaltung um 11:35 wurde durch das Einschalten eines weiteren Verbrauchers (Geschirrspüler) verursacht, erkennbar

durch den Anstieg des Stromverbrauches (orange Linie) sowie auch des anhaltenden Stromverbrauches nach der Abschaltung der Heizung. Das System hat erkannt dass, verursacht durch den zusätzlichen Verbraucher, nicht mehr genug Strom aus Eigenproduktion zur Verfügung steht, und hat die Heizung des Warmwassers abgeschaltet.

Nachdem der Geschirrspüler gegen 12:05 beendet wurde, kam es aufgrund fehlender Sonnenenergie, auch zu einem Abfall des produzierten elektrischen Stromes (verursacht durch Wolken). Erst nachdem gegen 12:45 wieder genug Strom produziert wurde, wurde die Heizung vom System wieder automatisch eingeschaltet. Die Heizung lief danach bis zum Zeitpunkt an dem die Solltemperatur des Warmwassers erreicht wurde.

## 3.4 Auszug aus dem Quellcode des Steuerungsprogrammes

```
time G_tOn;
bool G_bOn;

float G_fStartTemp;
float G_fStartPower;
float G_iStartDelay;
float G_fStopTemp;
float G_fStopPower;
float G_iStopDelay;

main()
{
  dpConnect("cbConfig",
            "Heater.Start.Temp",
            "Heater.Start.Power",
            "Heater.Start.Delay",
            "Heater.Stop.Temp",
            "Heater.Stop.Power",
            "Heater.Stop.Delay");

  dpConnect("cbHeater", true, "Heater.Control.Power");
}

void cbConfig(
    string dp1, float fStartTemp,
    string dp2, float fStartPower,
    string dp3, float iStartDelay,
    string dp4, float fStopTemp,
    string dp5, float fStopPower,
    string dp6, float iStopDelay)
{
  G_fStartTemp  = fStartTemp;
```

```
  G_fStartPower  = fStartPower;
  G_iStartDelay = iStartDelay;
  G_fStopTemp   = fStopTemp;
  G_fStopPower   = fStopPower;
  G_iStopDelay  = iStopDelay;
}


void cbHeater(string dp1, float power)
{
  time t = getCurrentTime(), tLastSet;

  float temp;

  dpGet("Heater.Control.OnOff", G_bOn,
        "Heater.Control.OnOff:_online.._stime", tLastSet,
        "Heater.Control.Temp", temp);

  int bOn=G_bOn;

  if ( G_bOn == false )
  {
    if ( temp <= G_fStartTemp && power > G_fStartPower && G_tOn < t-
60*G_iStartDelay )
    {
      bOn = true;
      G_tOn = t;
    }
  }
  else if ( G_bOn == true )
  {
    if ( temp >= G_fStopTemp || (power < G_fStopPower && G_tOn < t-
60*G_iStopDelay) )
      bOn = false;
  }

  if ( G_bOn != bOn || tLastSet < t-60*5 )
  {
    DebugTN("state set "+bOn);
    dpSet("Heater.Control.OnOff", bOn);
    tLastSet=t;
    G_bOn=bOn;
  }

  string s;
  sprintf(s, "state=%d %s temp=%7.2f / %5.2f power=%5.0f", (int)G_bOn,
(string)G_tOn, temp, temp*0.5-297.5, power);
  DebugTN("cbHeater", s);
}
```

## 3.5 Monitoring der Systemkomponenten und Fernwirken

Für das Monitoring der Systemkomponenten und zum Fernwirken wurde zusätzlich zum SCADA-System eine Mobile App entwickelt.

Mit dieser können Photovoltaikwerte, Zählerwerte, Temperaturwerte, Heuristikwerte, usw. visualisiert werden. Die im System integrierten Schalter können manuell ein- und ausgeschalten werden und die automatisierte Steuerung des Heizstabes kann aktiviert und deaktiviert werden.


Abb. 19) Mobile-App Bedienoberfläche für die Steuerung und Trenddarstellung

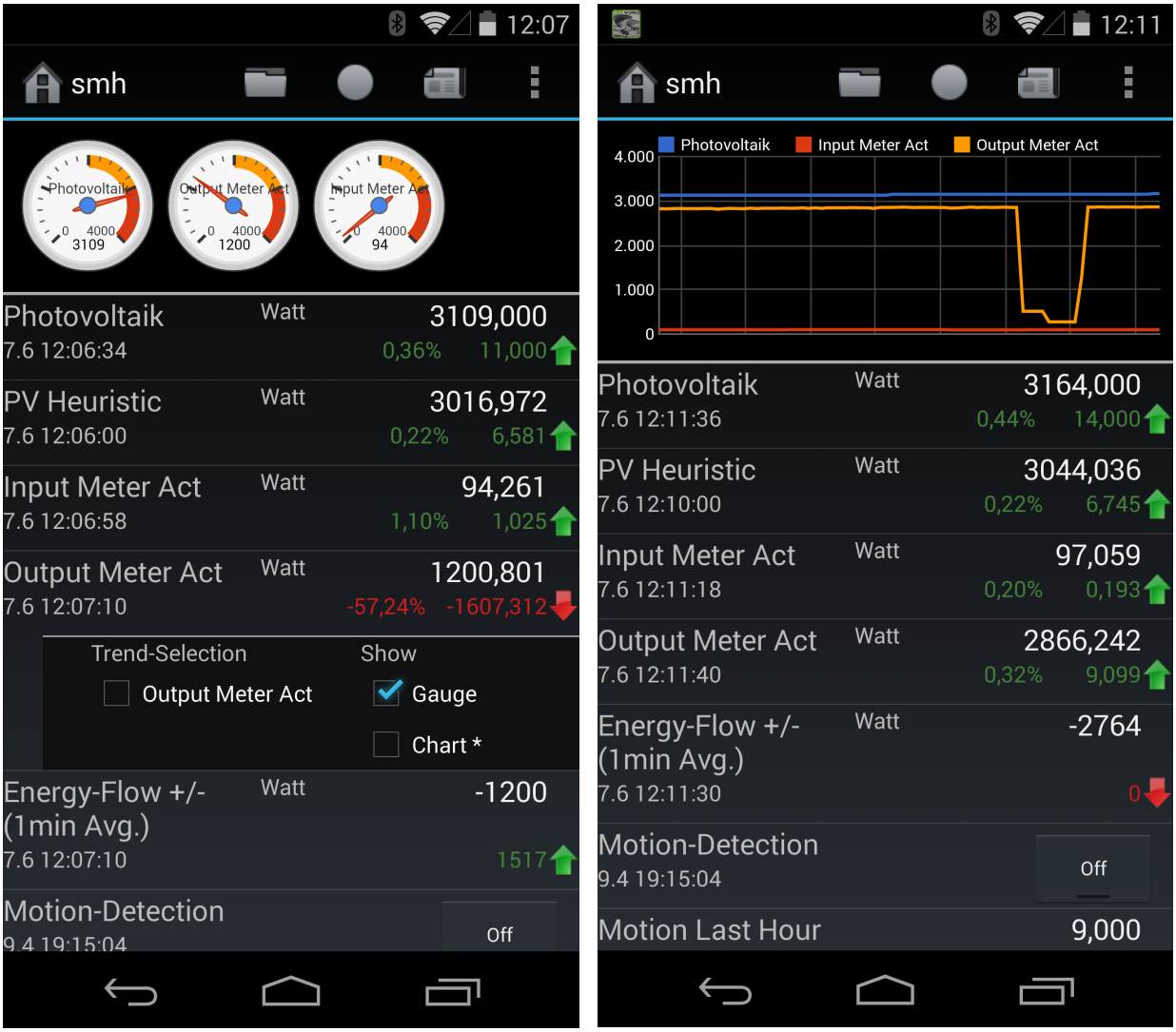



Abb. 20) Mobile-App Bedienoberfläche für die Visualisierung und integrierter Trenddarstellung

Die Mobile-App soll aufzeigen dass das Montoring sowie auch ein Fernwirken über die Anbindung einer Mobile-App mit dem implementierten System möglich ist. Es wird jedoch in der vorliegenden Arbeit nicht weiter darauf eingegangen.

# 4 Zusammenfassung

Die vorliegende Arbeit zeigt die Umsetzung eines Regelungs- und Steuersystems zur Eigenverbrauchsoptimierung von elektrischem Strom aus Eigenproduktion. Der Eigenverbrauch wird durch Automatisierung der Heizung eines Warmwasserboilers in Abhängigkeit von produziertem Photovoltaikstrom gesteuert.

Durch die zunehmende Verbreitung von stromproduzierenden Anlagen (vornehmlich Photovoltaikanlagen) nehmen die Vergütungen für in das Stromnetz eingespeisten Strom ab. Der Ertrag von eigenerzeugtem Strom liegt unter dem Preis von zugekauftem Strom. Durch die Erhöhung des Eigenverbrauches mittels automatisierter Steuerung kann die Wirtschaftlichkeit einer Photovoltaikanlage entscheidend verbessert werden.

Aufgrund langer Produktzyklen erweisen sich Firmen im Steuerungs- und Regelungsbereich als vergleichsweise innovationsscheu und teuer. Doch mit Systemen wie Arduino und Raspberry Pi, die einen einfachen Einstieg in die Embedded Programmierung und Regelungs- und Steuertechnik ermöglichen, kann ein intelligentes Steuer- und Regelungssystem zur Eigenverbrauchsoptimierung von Photovoltaikstrom durch thermische Speicherung in Einfamilienhäusern ohne hohe Investitionskosten umgesetzt werden.

Aufbauend auf den Erkenntnissen der Bachelorarbeit 1 „Eigenverbrauchsoptimierung von Photovoltaikstrom in Einfamilienhäusern“ (Vogler, 2014), werden in der Arbeit die Teilkomponenten, um ein derartiges Regelungs- und Steuerungssystem zu erstellen, umgesetzt. Diese Teilsysteme werden danach in ein Gesamtsystem zur Regelung und Steuerung integriert.

Das Ergebnis ist ein System welches durch die automatisierte Steuerung eines elektrischen Heizstabes einen thermischen Speicher, in Form eines Warmwasserboilers, heizt. Das Aufheizen erfolgt nur, wenn Strom aus Eigenproduktion im Überfluss zur Verfügung steht.

Die Teilkomponenten wurden als eigenständige Komponenten umgesetzt, welche über Serviceschnittstellen mit dem Gesamtsystem kommunizieren. Diese lose Verbindung, durch die Verwendung von Services, führt zu einer hohen Verteilbarkeit der Teilsysteme, und richtet sich dem aktuellen IT-Trend von „IoT – Internet of Things“.

Ein Aspekt des Lösungsweges sind die Investitionskosten. Diese wurden gering gehalten, sodass die Amortisation des Investments der Anlage und des Steuerungssystems nicht den Nutzen des Eigenverbrauches wesentlich mindert. In

dieser Arbeit wurde daher auf teure industrielle und kommerzielle Komponenten so weit wie möglich verzichtet.

Eingesetzte kommerzielle Komponenten (Oracle Datenbank, SCADA-System) waren notwendig, da diese zur Findung eines passenden Lösungsweges einen breiteren Bereich an Funktionalitäten bieten. Mit diesen Funktionalitäten können beschrittene Lösungswege schneller auf dessen Zielerreichung überprüft werden. Für die umgesetzte Lösung könnten in einem weiteren Schritt die kommerziellen Komponenten durch alternative und frei verfügbare Komponenten ersetzt werden. Durch die Integration der Teilsysteme über Serviceschnittstellen ist der Austausch von Teilkomponenten möglich, ohne abhängige Komponenten zu verändern.

Als Nebenprodukt wurde, durch die für die Steuerung notwendige Erfassung von Daten über die Stromproduktion und dem Stromverbrauch, eine Visualisierung geschaffen. Auch wenn ein nachhaltiges Ändern des Verbrauches aufgrund der Unregelmäßigkeit des produzierten Stromes schwer bzw. nur in bedingtem Maße und in bestimmen Bereichen möglich ist, so kann durch die Visualisierung zumindest das Bewusstsein des Konsumenten über die Kosten und des Ertrages geschärft werden. Die automatisierte Steuerung von regelmäßig notwendigen Verbrauchern erfolgt durch das umgesetzte Regelungs- und Steuerungssystem.

Bereits in der Bachelorarbeit 1 wurde in einem Laborversuch über einen Zeitraum das Aufheizen des thermischen Warmwasserspeichers mit dem geschaffenen Regelungs- und Steuerungssystem durchgeführt. Durch die Auswertung der aufgezeichneten Messdaten konnte verifiziert werden, dass das System zum gewünschten Ziel „Der automatisierten Steuerung von Endgeräten zur Erhöhung des Eigenverbrauches von selbst erzeugtem Strom“ führt.

Zusammen mit den in der in der Bachelorarbeit 1 durchgeführten Laborversuchen konnte damit die Umsetzbarkeit und die Leistungsfähigkeit einer kostengünstigen Anlage zur Eigenverbrauchsoptimierung von Photovoltaikstrom in Einfamilienhäusern nachgewiesen werden.

Die Übersetzung dieser Musterlösung in ein entsprechendes kostengünstiges und verkäufliches Produkt ist ein Ziel weiterer Arbeiten.

# 5 Literaturverzeichnis

Vogler, A. (2014): Eigenverbrauchsoptimierung von Photovoltaikstrom in Einfamilienhäusern. Bachelorarbeit 1, FH-Burgenland University of Applied Sciences.

Kalt, G. & Baumann, M (2013): Modellierung zukünftiger Entwicklungen des Haushaltsstromverbrauchs im Kontext von Time-of-use-Tarifen, Load-shiftig, Elektromobilität und Energieeffizienz: Internationale Energiewirtschaftstagung 2013 an der TU Wien, Wien.

Hoberg, A; Piele, C. & Veit, J. (2013): Mobiles Lernen für Smart Home/Smart Grid. HMD Praxis der Wirtschaftsinformatik, Heft 291, Sonderdruck.

Westermann, D.; Döring, N. & Bretschneider P. (Hrsg.) (2013): Smart Metering. Zwischen technischer Herausforderung und gesellschaftlicher Akzeptanz – Interdisziplinärer Status Quo, Band 4, Universitätsverlag Ilmenau.

Müller K.; (2010): Sicherheit in vernetzten Systemen, Workshop, Firma Secorvo Security Consulting GmbH, Karlsruhe.

Hillemacher, L.; Eßer-Frey, A. & Fichtner, W. (2011): Preis- und Effizienzsignale im MeRegio SmartGrid Feldtest – Simulationen und erste Ergebnisse, Internationale Energiewirtschaftstagung 2011 an der TU Wien, Wien.

Bost, M.; Hirschl, B. & Aretz A. (2011): Effekte von Eigenverbrauch und Netzparität bei der Photovoltaik. Studie im Auftrag von Greenpeace Energy eG, Langfassung, Institut für ökologische Wirtschaftsforschung.

Huber, M.; Sänger, F. & Hamacher T. (2013): Das „Post-EEG"-Potenzial von Photovoltaik im privaten Strom- und Wärmesektor. Energiewirtschaftliche Tagesfragen, Jg. 2013 Heft 9

Wachenfeld V.; Engel, B. & Rothert, M. (2009): Leistungselektronik von PV Anlagen nach Erreichen der „Grid Parity", Conference Paper, Internationaler ETG Kongress 2009

Hanna T. (2014): Embedded-Programmierung im Umbruch. http://www.heise.de/developer/artikel/Embedded-Programmierung-im-Umbruch-2082301.html, abgerufen am 05.04.2014 10:00 Uhr

Raspberry Pi (2014): www.raspberrypi.org, abgerufen am 30.05.2014, 17:00 Uhr

Arduino (2014): www. arduino.cc, abgerufen am 30.05.2014, 17:00 Uhr

SMA Hersteller (2014): www.sma.de, abgerufen am 13.02.2014, 20:00 Uhr

SMA Web-Portal (2014): www.sunnyportal.com, abgerufen am 13.02.2014, 20:00 Uhr

SMA Spot (2014): Software zum Auslesen der SMA Bluetooth-Schnittstelle, https://code.google.com/p/sma-spot/, abgerufen am 13.02.2014, 20:00 Uhr

SMA Bluetooth (2014): Software zum Auslesen der SMA Bluetooth-Schnittstelle, https://code.google.com/p/sma-bluetooth/wiki/Installation, abgerufen am 13.02.2014, 20:00 Uhr

The Pi4J Project (2014): http://pi4j.com, abgerufen am 05.04.2014, 21:45 Uhr

Philips KTY81-2 (2014): http://www.produktinfo.conrad.com/datenblaetter/150000-174999/153653-da-01-en-TEMPERATUR_SENSOR_KTY10_7_KTY81_222.pdf, Produktinformation Conrad.t, abgerufen am 06.04.2014, 09:00 Uhr

GPRS General Packet Radio Services (2014): http://en.wikipedia.org/wiki/General_Packet_Radio_Service

Oracle Database Express Edition (2014): http://www.oracle.com/technetwork/database/database-technologies/express-edition, abgerufen am 13.02.2014, 20:00 Uhr

Oracle Application Express (2014): https://apex.oracle.com, abgerufen am 30.05.2014, 17:30 Uhr

SIMATIC WinCC Open Architecture (2014): http://www.automation.siemens.com/mcms/human-machine-interface/de/visualisierungssoftware/simatic-wincc-open-architecture/Seiten/Default.aspx, abgerufen am 05.04.2014, 21:45 Uhr

SIMATIC WinCC Open Architecture Managerkonzept (2014): http://www.etm.at/index.asp?id=2&sb1=58&sb2=115, abgerufen am 30.05.2014, 17:30 Uhr

ELRO Funk Steckschalter (2014): http://www.elro.eu/de/produkte/cat/home-automation/home-easy-next/empfaenger-on-off2, abgerufen am 13.02.2014, 20:00 Uhr

ELRO-USB-Dongle-Modul (2014): http://www.elro.eu/de/produkte/cat/home-automation/home-easy-next/sender2/fernbedienbarer-pc-usb-dongle, abgerufen am 13.02.2014, 20:00 Uhr

DIN EN 62053-31 (2014): http://www.dke.din.de/cmd?artid=11912697&subcommitteeid=54758653&bcrumblevel=1&contextid=dke&level=tpl-art-detailansicht&committeeid=54738887&languageid=de, abgerufen am 01.06.2014, 9:30 Urh

Wikipedia Pulsweitenmodulation (2014): http://de.wikipedia.org/wiki/Pulsweitenmodulation

# EIDESSTATTLICHE ERKLÄRUNG

Hiermit erkläre ich ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

---------- ----------

Ort, Datum Unterschrift