Masterstudiengang

„Telekommunikation und Medien“

Prozessdaten im WWW

Welche Möglichkeiten Prozessdaten aus der Feldebene auf dem Weg zu einem Webserver haben

Betreut von

DI Grischa Schmiedl

Verfasst von

Christoph Windl

tm071069

St. Pölten im März 2009

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter oder einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der von den Begutachter/inne/n beurteilten Arbeit überein.

.....	.....
Ort, Datum	Unterschrift

Zusammenfassung

Der Zugriff auf Prozessdaten in der Produktionsebene wird immer wichtiger. Sei es zur Beweisführung (Datalogging), Wartung (Störmeldesystem) oder Steuerung (Prozessleitsystem) der Anlage. Fertige Produkte der Hersteller sind unflexibel und/oder kostenintensiv. Sobald Geräte verschiedener Hersteller zum Einsatz kommen, führt dies im Normalfall zu noch größerem Aufwand.

Als Alternative wäre eine Webanwendung zur Überwachung und Steuerung von Automatisierungsgeräten denkbar, da Webbrowser heutzutage bereits für die vielfältigsten Aufgabenbereiche eingesetzt werden. Aber auch hier ist ein Zugriff auf die Prozessdaten der Automatisierungsgeräte notwendig.

Durch den Einsatz von Ethernet als Feldbus in der Automatisierungstechnik ist zumindest der Zugriff auf Prozessdaten der physikalischen Ebene erheblich erleichtert worden, weil übliche IT-Hardware verwendet werden kann. Muss auf einen bestehenden, nicht Ethernet-basierten Feldbus zugegriffen werden, benötigt man spezielle Schnittstellen-Hardware.

Jeder Hersteller von Automatisierungsgeräten verwendet zur Kommunikation seine eigenen Protokolle. Für den Zugriff auf die Prozessdaten wird ein passender Treiber für sein Protokoll benötigt. Müssen Geräte von verschiedenen Herstellern angesprochen werden, dann empfiehlt sich ein OPC-Server, welcher eine standardisierte Schnittstelle zu allen Geräten der OPC-Mitgliedsfirmen bietet. Der Zugriff auf diese Schnittstelle erfolgt mithilfe eines OPC-Clients.

Alternativ kann auch ein Linux-Feldbus-Controller eingesetzt werden. Der eingesetzte Linux-Kernel wurde speziell für den Einsatz mit Mikrocontrollern ohne Speicherverwaltungseinheit entwickelt. Man darf sich zwar nicht den Komfort von einem Desktop-Linux erwarten, aber viele bekannte Programme sind auch für diese uClinux genannte Distribution verfügbar.

Mithilfe des Linux-Feldbus-Controllers wurde ein Projekt umgesetzt, dessen Vorgabe die Erstellung eines Störmeldesystems war. Dabei werden Informationen über Störmeldungen von Geräten und Maschinen über digitale Eingangsklemmen aufgenommen. Diese Daten werden an einen Apache-Webserver übermittelt und in einer MySQL-Datenbank abgelegt. Diese Informationen werden mithilfe eines Webbrowsers abgerufen und mit Javascript aktuell gehalten. Mithilfe der Quittierungsfunktion kann der Anwender den Zustand von aktiven Störmeldungen verändern.

Abstract

Access to process data in the production level is becoming increasingly important. Process data are used for evidence (data logging), maintenance (fault indicating system) or control (process control) of the plant. Commercial products are inflexible and / or cost-intensive. Devices used from different manufacturers are costly in terms of labour.

An alternative is a web application to monitor and control automation devices as web browsers today already used for various tasks. But it is essential to access to the devices, too.

The use of Ethernet as a fieldbus in automation technology has become much easier because common IT hardware can be used. Accessing to an existing non-Ethernet-based fieldbus requires special hardware interfaces.

Each manufacturer of automation devices uses its own protocol for communication. A suitable driver for a specific protocol is required to access to process data. If communication between devices from different manufacturers is needed an OPC server is recommended. It is a standardized interface to all devices of the OPC member companies. The access to this interface is established by an OPC client.

Alternatively a Linux fieldbus controller can be used. The Linux kernel is developed specifically for using micro controllers without a memory management unit. A Linux fieldbus controller has not the comfort of a desktop Linux, but many well-known programs are also available for this distribution, called uClinux.

Using the Linux fieldbus controller, a project was implemented, whose target is the creation of a fault indicating system. It provides malfunction information on devices via digital input terminals. The malfunction information are sent to an Apache webserver and stored in a MySQL database. A web browser indicates the received data, Javascript keeps them up-to-date. The user can change the status from “not OK” to “acknowledged”.

Inhaltsverzeichnis

1 Einleitung....1
1.1 Motivation....2
1.2 Ziel der Arbeit....3
1.3 Wichtiger Hinweis....3
1.4 Prozess- bzw. Feldkommunikation....4
1.5 Horizontale Kommunikation....4
1.6 Vertikale Kommunikation....5
2 Varianten des Zugriffs auf Prozessdaten....6
2.1 OPC....6
2.1.1 OPC-Server....8
2.1.2 OPC-Client....8
2.1.3 OPC-Spezifikationen....9
2.2 Feldbus....14
2.2.1 Industrial Ethernet....14
2.2.2 PROFINET....15
2.3 Linux Controller von Wago....17
2.3.1 I/O-System 750 von Wago....17
2.3.2 Linux Systemstart....20
2.3.3 Verbindungsaufbau mit der Konsole des Feldbus-Controllers....22
2.3.4 Das Dateisystem des Feldbus-Controllers....23
2.3.5 Web-based Management des Feldbus-Controllers....27
2.3.6 Prozessabbild am Feldbus-Controller....28
2.3.7 Beispielprogramme am Feldbus-Controller....33
3 Projekt....35
3.1 Aufgabenstellung....35
3.2 Gesamtkonzept....36
3.2.1 Webbrowser....37
3.2.2 Webserver....37
3.2.3 Feldbus-Controller....38
3.3 Testaufbau....39
3.3.1 Die Hardware des Testaufbaus....40
3.3.2 Client-Rechner....48
3.3.3 Webserver....54
3.3.4 Feldbus-Controller....60

3.4 Probleme bei der Projektumsetzung.....65
3.5 Mögliche Erweiterungen.....67
4 Fazit.....68
5 Anhang.....69
5.1 Anhang A: Literaturverzeichnis.....69
5.1.1 Bücher.....69
5.1.2 Zeitschriften.....69
5.1.3 Handbücher und Datenblätter.....70
5.1.4 Bachelor's Thesis.....70
5.1.5 Normen.....70
5.1.6 Katalog.....70
5.1.7 Präsentationen.....71
5.1.8 Bilder.....71
5.2 Anhang B: Quellenangaben aus dem Internet.....72
5.3 Anhang C: Abbildungsverzeichnis.....73
5.4 Anhang D: Tabellenverzeichnis.....74
5.5 Anhang E: Listingverzeichnis.....75
5.6 Anhang F: Danksagungen.....77

1 Einleitung

Industrielle Anlagen bzw. Maschinen erledigen ihre Arbeit immer selbstständiger und auch ihre Aufgabengebiete werden immer komplexer. Aufgrund dieser Selbständigkeit benötigen diese Maschinen immer weniger Mithilfe durch Personal und die Produktionskosten sinken. Je nach Anwendungsgebiet können sich weitere Vorteile wie z.B. Steigerung der Produktionsmenge, Steigerung der Produktionsqualität oder körperliche Entlastung des Personals ergeben. Der Aufwand für Service und Wartung fällt im Gegensatz zum Nutzen solcher Anlagen in der Regel eher gering aus. Für die Planung, Entwicklung und den Betrieb solcher Anlagen ist das Ingenieurwesen der Automatisierungstechnik zuständig.

Das zentrale Element zur Steuerung solcher Anlagen bildet dabei eine speicherprogrammierbare Steuerung, kurz SPS. Über Geber bzw. Sensoren wird der aktuelle Zustand (Position, Temperatur, Druck, Menge, ...) der Anlage erfasst. Ein Programm in der SPS, welches für eine bestimmte Aufgabenstellung entwickelt wird, benötigt diese Eingangsdaten, auch Prozessabbild genannt, zur Berechnung weiterer Aktionen. Diese Aktionen werden über die Ausgänge der SPS an die Aktoren (Motor, Ventil, Heizung, ...) der Maschine weitergegeben.

Eine wesentliche Schnittstelle der Automatisierungstechnik ist die Bedienung. Über diese Mensch-Maschine-Schnittstelle (Human-Machine-Interface, kurz HMI) wird dem Bedienpersonal die Steuerung der Anlage (Start, Stop, NOT-AUS, Weiter, …) ermöglicht. Ein bekanntes Beispiel eines HMI ist das Armaturenbrett in einem KFZ. Die Bedienung kann bei einfacheren Systemen nur über Taster, Schalter und Meldeleuchten erfolgen. Bei komplexeren Automatisierungssystemen werden zusätzlich zu den mechanischen Bedienelementen Visualisierungssysteme, auch Bedienterminal genannt, verwendet.

Die ersten Bedienterminals (Panels) waren rein textbasiert. Einfache Bedienterminals sind auch heute noch zum Teil textbasiert ausgeführt. Außerdem sind sie kostengünstig und besitzen kleine Abmessungen. Inzwischen gibt es eine große Auswahl an verschiedenen Geräteklassen: Kleine textbasierte Panels, Mobile mit WLAN ausgerüstete Panels, Grafische Panels, Panels mit Touchscreen, Panel-PCs, …

Abbildung 1: Verschiedene Ausführungen von Bedienterminals (von links nach rechts): Textbasiert (Siemens AG 2009a: Text Display TD 400C), Mobil inkl. Touchscreen (Siemens AG 2009b: Mobile Panel 277F IWLAN) und Grafik inkl. Touchscreen (Siemens AG 2009c: THIN CLIENT PRO 15 Zoll mit Erweiterungseinheiten)

Diese Geräte tragen aber auch mehr oder weniger ihren Teil zu den Gesamtkosten einer Anlage bei. Dies variiert natürlich je nach Ausführung und Anzahl der benötigten Geräte. Zusätzlich müssen auch diese Geräte entsprechend der Vorgaben des Kunden programmiert werden. Auch die Kosten für die Integration in das Gesamtsystem inkl. erforderlichem Anschluss an das Kommunikationsnetz der Anlage dürfen nicht übersehen werden.

1.1 Motivation

Webbrowser sind Standard-Programme für normale PC's, die zur Visualisierung von Informationen auf Webseiten dienen. Dabei spielt es keine Rolle um welche Daten es sich handelt, da diese bereits vom Webserver aufbereitet werden. Auch der Art der Darstellung sind kaum Grenzen gesetzt. Ob einfach und übersichtlich oder bunt und überladen, die Art der Darstellung liegen in der Hand des Kunden und des Programmierers. Ein Webbrowser wäre also auch prädestiniert für die Visualisierung von Prozessdaten.

Üblicherweise wird dafür aber eine nur für diesen Zweck entwickelte und teure Hardwarelösung eingesetzt, die sogenannten Bedienterminals. Diese sind im Normalfall ortsgebunden und für performante und flexibel einsetzbare Geräte benötigt man meist auch noch kostenpflichtige Softwarelizenzen.

Das Problem für einen Einsatz von Webbrowsern für diese Aufgabe liegt vielmehr in der Art der Prozessdaten, den proprietären Protokollen und dem Transport der Daten vom

Automatisierungsgerät zum Webserver. Jeder Hersteller dieser Systeme verwendet sein eigenes Format für die Prozessdaten und für den Zugriff auf diese Daten wird fast immer zusätzliche Hardware benötigt. Je nach Anwendungsfall muss auch noch eine mehr oder weniger große Wegstrecke für den Datentransport zwischen Automatisierungssystem und dem Webserver oder einer Datenbank überbrückt werden.

Web basierende HMI- oder SCADA-Systeme (Supervisor, Control And Data Acquisition) für PC's sind zwar am Markt verfügbar, diese Softwarepakete sind aber sehr komplex in der Anwendung und kostenintensiv in der Anschaffung. Der Einsatz lohnt sich daher nur für große Projekte. Bei der Programmierung muss auf die zur Verfügung gestellten Bibliotheken Rücksicht genommen werden. Die Logik für die Steuerung der Anlage verbleibt bei diesem Konzept aber nach wie vor auf einem Automatisierungssystem.

(vgl. Langmann 2007, S. 5 ff.)

1.2 Ziel der Arbeit

Es werden Wege gesucht um Prozessdaten von Automatisierungssystemen aus der Feldebene (z.B. Produktionsbereich) im Webbrowser visualisieren zu können. Die größte Hürde hierbei, weil zeit- und kostenintensiv, die Prozessdaten nicht nur in ein anderes benötigtes Format zu wandeln, sondern auch einen durchgängigen Weg dieser Daten vom Automatisierungssystem bis zum Webserver zu schaffen.

Diese Arbeit behandelt jedoch nicht die grundsätzliche Erstellung von Websites, deren Protokolle und Programmier- bzw. Skriptsprachen. Es werden zwar auch aktuelle Webtechnologien erwähnt und eingesetzt, eine Übersicht über aktuelle Webtechnologien ist aber nicht Gegenstand dieser Arbeit.

1.3 Wichtiger Hinweis

SPS-Herstellerfirmen verwenden ähnliche Begriffe oft für komplett unterschiedliche Bedeutungen. In der Literatur wird der Begriff „embedded System“ für Systeme mit Microcontroller verwendet, welche eigene Betriebssysteme besitzen. Bei der Firma Siemens gibt es auch einen „embedded PC“ auf dem allerdings ein „Windows XP embedded“ vorinstalliert ist.

In der Literatur hingegen werden oft für die selbe Aussage unterschiedliche Bezeichnungen verwendet. Während so manche Literatur die Kategorie „SPS“ in Microcontroller-basierend und PC-basierend aufteilt, wird in einer anderen Literatur die Kategorie „Automatisierungsgerät“ in Hardware-SPS und PC-SPS unterteilt.

1.4 Prozess- bzw. Feldkommunikation

Diese Kommunikationsart dient dem Datenaustausch zwischen dem Automatisierungssystem (SPS) und den Sensoren und Aktoren (Master/Slave-Kommunikation). Für diese Kommunikation wird üblicherweise ein nicht ethernet-basierter, zyklischer Feldbus (PROFIBUS, AS-i-Bus, CAN-Bus, ...) eingesetzt.

(vgl. Wellenreuther et al. 2008, S. 505 f.)

(vgl. Autor unbekannt 5. 4. 2009, 1, Feldbusse im Vergleich, http://www.feldbusse.de/Vergleich/vergleich.shtml)

1.5 Horizontale Kommunikation

Große automatisierte Anlagen werden üblicherweise in Teilbereiche aufgeteilt. Auch die Automatisierungsgeräte werden auf diese Teilbereiche aufgeteilt. Um eine durchgängige Funktion der gesamten Anlage zu gewährleisten, benötigen zumindest die benachbarten Anlagenteile eine Kommunikationsmöglichkeit. Diese Art der Kommunikation wird als horizontale Kommunikation bezeichnet (Master/Master-Kommunikation) Hierfür wird wahlweise ein Ethernet-basierter oder nicht Ethernet-basierter Feldbus verwendet. Der Datenaustausch geschieht jedoch asynchron. Unter die Bezeichnung „Horizontale Kommunikation“ fällt auch der Datenaustausch mit Bediengeräten.

(vgl. Wellenreuther et al. 2008, S. 704)

(vgl. Langmann. 2004, S. 334 f.)

1.6 Vertikale Kommunikation

Bei der vertikalen Kommunikation handelt es sich um den Datenaustausch zwischen Produktionsebene und Verwaltungsebene. Das Hauptaugenmerk liegt dabei nicht auf der Datenübertragung, sondern auf der Verständigung der beiden Ebenen weil jeder Hersteller von Automatisierungsgeräten seine „eigene Sprache“ spricht. Hat man dieses Problem erst durch den Einsatz herstellerspezifischer Treiber gelöst, so kommt mittlerweile immer öfter die OPC-Schnittstelle (siehe Kapitel 2) zum Einsatz.

Die Verfügbarkeit von Ethernet-basierten Feldbussen hat der vertikalen Kommunikation zu vermehrtem Einsatz verholfen, da dadurch die Anschaffung von spezieller Kommunikations-Hardware entfällt. Der Übergang vom Informationsbus (Intranet) auf den Systembus (Feldbus) wird somit erleichtert. Auch lassen sich PC-basierte Automatisierungssysteme leichter in die normale IT-Infrastruktur integrieren und bei Wartungsarbeiten über das Internet steht somit nichts mehr im Weg.

Alternativ bildet auch die Prozessleittechnik, in Form eines industriellen SCADA-Softwarepakets, die Schnittstelle zur materiellen Produktion und ist damit Teil der prozessnahen Informationstechnik. Sie liefert Informationen vom Produktionsprozess und kann Vorgaben aus dem Managementbereich in den Produktionsbereich einbringen. Auch hier wird ein Trend hin zu IT-Standards beobachtet. Die Rezeptsteuerung ist bei der Prozessleittechnik ein bekannter und verbreiteter Begriff. Über die Rezeptsteuerung kann z.B. die Zusammensetzung eines Produktes von einem Büroarbeitsplatz aus mithilfe eines Microsoft Excel Add-Ins geändert werden. Obwohl in einem Unternehmen grundsätzlich alles mit allem zusammenhängt, ist der Weg hin zu einer vollständigen informationstechnischen Integration sämtlicher Unternehmensfunktionen noch in weiter Ferne (Stichwort „die vernetzte Fabrik“).

(vgl. Früh et al. 2008, S. 29 ff.)

(vgl. Wellenreuther et al. 2008, S. 6 ff.)

(vgl. Langmann 2004, S. 334 f.)

(vgl. Autor unbekannt 18. 3. 2009, 1, OPCEx Excel Add-In, http://www.resolvica.com/P1/default.aspx)

2 Varianten des Zugriffs auf Prozessdaten

In diesem Kapitel werden unterschiedliche Arten des Zugriffs auf Prozessdaten beschrieben. OPC benötigt ebenfalls einen Feldbus um mit den Automatisierungsgeräten kommunizieren zu können. Im Gegensatz zu einem Feldbus-Treiber stellt OPC für den Zugriff auf Prozessdaten eine standardisierte Schnittstelle für Geräte unterschiedlicher Hersteller zur Verfügung. Mit diesen beiden Varianten könnten auch bestehende Automatisierungsanlagen erweitert werden.

Die Variante mit dem Feldbus-Controller setzt an einem anderen Punkt an. Die Prozessdaten werden direkt im Automatisierungsgerät für die Weiterverarbeitung, zum Beispiel auf einem Webserver, aufbereitet. Ein Feldbus wird für den Zugriff auf Prozessdaten nicht mehr benötigt. Bestehende Automatisierungsgeräte müssten allerdings ersetzt werden. Auch die Software kann nicht einfach übernommen werden, weshalb sich diese Variante nur für neue Automatisierungsanlagen rentiert.

2.1 OPC

Um auf die Prozessdaten einer SPS zugreifen zu können benötigt man einen dafür angepassten Treiber. Möchte man auf dieselben Daten einer SPS eines anderen Herstellers zugreifen, funktioniert das allerdings nicht mit dem selben Treiber. Die Treiber sind nur für ein bestimmtes Fabrikat zuständig. Visualisierungssysteme benötigen daher oft mehrere Treiber um auf alle benötigten Prozessdaten zugreifen zu können.

OPC bietet eine standardisierte, herstellerunabhängige Schnittstelle zwischen Anwendersoftware und Prozessdaten verschiedenster Automatisierungssysteme. OPC bildet also herstellerspezifische Daten auf eine allgemein gültige, standardisierte Schnittstelle ab. Ursprünglich stand OPC für „OLE for Process Control“ und OLE wiederum für „Object Linking and Embedding“. OLE wurde durch das Component Object Model (COM) und Distributed COM (DCOM) abgelöst. Seit der Erweiterung einiger OPC-Spezifikationen um XML und Web-Services, steht OPC heute für „Openess, Productivity and Collaboration“ und somit nicht mehr für eine zugrunde liegende Technologie.

Der Grund für die Entwicklung von OPC liegt in der geforderten Verschmelzung von Bürobereich mit Automatisierungsbereich. Durch die Einführung des defakto Büronetzwerkstandards „Ethernet“ in der Automatisierungstechnik ist das Ziel der Vernetzung zwischen Büro und Automatisierung wesentlich leichter umsetzbar geworden. In der Automatisierungstechnik wird dieses Feldbussystem allerdings als „Industrial Ethernet“ bezeichnet.

Diese Integration soll dem Bürobereich folgende Möglichkeiten in der Verwaltung der Automatisierung bieten:

- Bedienung und Beobachtung
- Störungserkennung
- Auswertung und Archivierung von Prozessdaten
- Optimierung des Produktionsablaufs
- Integrierte Betriebsführung (Qualitätskontrolle, Instandhaltung, Materialverwaltung, Auftragswesen und Produktionsplanung)

Die Kommunikation beruht auf dem Client-Server-Prinzip. Dabei bietet der Server dem Client seine Dienste an. Sendet der Client eine Anfrage an den Server führt dieser die Aufgabe aus und sendet eine Antwort zurück an den Client.

(vgl. Wellenreuther et al. 2008, S. 7 ff.)

(vgl. Lange 2005, S. 64)

(vgl. Autor unbekannt 9. 4. 2009, 1, About OPC – What is OPC?, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC#open)

2.1.1 OPC-Server

OPC-Server werden vom jeweiligen Hersteller der eingesetzten Automatisierungssysteme angeboten. Diese Software wird grundsätzlich für „normale“ x86-Computer entwickelt und der Computer auf dem diese Software eingesetzt wird benötigt in der Regel noch eine herstellerspezifische Kommunikationsmöglichkeit zu dem Automatisierungssystem. Dies erfolgt grundsätzlich über einen Feldbus (PROFINET, Ethernet/IP, ModbusTCP, …), wofür eventuell eine zusätzliche Hardwarekomponente zur physikalischen Verbindung mit dem Computer benötigt wird. Über dieses Kommunikationsnetz ist der OPC-Server mit den Automatisierungsgeräten verbunden, dessen Prozessdaten der Server auf seiner OPC-Schnittstelle zur Verfügung stellt. Bei einer Anfrage vom Client wird auf die Daten des betreffenden Automatisierungsgeräts zugegriffen. Jeder OPC-Server besitzt deshalb einen lesbaren Namen und eine ID mit 32 Stellen zur eindeutigen Identifizierung.

(vgl. Wellenreuther et al. 2008, S. 7 ff.)

2.1.2 OPC-Client

Um auf die Daten eines OPC-Servers zugreifen zu können benötigt man einen OPC-Client, welcher vor dem Einsatz erst konfiguriert werden muss. Bei dieser Konfiguration kann zwischen zwei Anwender-Schnittstellen gewählt werden:

- Custom-Interface: für Programmiersprachen wie z.B. C/C++, welche das Funktionszeiger-Prinzip nutzen können.
- Automation-Interface: für Programmiersprachen wie z.B. VB, die Schnittstellen mit Objektnamen (COM bzw. DCOM) ansprechen. Für diese Konfiguration ist allerdings eine zusätzliche DLL vom OPC-Server-Hersteller erforderlich.

Ein auf COM- oder DCOM-basierender OPC-Client kann zum Beispiel mit Excel-VBA bzw. als ActiveX-Objekt erstellt werden.

Ein OPC-Client kann auch einem Webserver die Prozessdaten zur Verfügung stellen. Dadurch können diese Daten einem Webbrowser zugänglich gemacht werden.

(vgl. Wellenreuther et al. 2008, S. 7)

2.1.3 OPC-Spezifikationen

Die OPC Foundation hat seit ihrer Gründung inzwischen einige Spezifikationen hervorgebracht. Bis vor wenigen Jahren wurde auf die COM/DCOM-Technologie von Microsoft gesetzt. Inzwischen werden Webservices genutzt, um eine Plattform-Unabhängigkeit zu erreichen. Trotzdem werden bestehende Spezifikationen noch weiterentwickelt. Nachfolgend werden einige Spezifikationen erläutert.

(vgl. Autor unbekannt 9. 4. 2009, 1, About OPC – What is OPC?, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC#open)

OPC DA (Data Access)

„Data Access“ ist die erste freigegebene OPC-Spezifikation. Sie dient dem Austausch von Echtzeitdaten zwischen Automatisierungsgeräten und Visualisierungsgeräten. Es ist Wichtig zu beachten, dass OPC DA Schnittstellen definiert, welche für den Prozessdatenaustausch zwischen Windows-basierten Anwendungen aufgrund der COM/DCOM-Technologie von Microsoft spezifiziert wurde. Der neuere Standard OPC XML-DA ist nicht nur Betriebssystem unabhängig, sondern auch Internet-fähig. Die zukünftige Version 3 von DA kann auch mit dem XML-DA Schema umgehen.

Mithilfe eines Projektierungstools muss eine Objekthierarchie erstellt werden. Oben beginnend setzt sich diese aus folgenden Ebenen zusammen:

- das oberste OPC-Objekt ist das „OPC-Server“-Objekt. Darin werden die Group-Objekte verwaltet. Das „OPC-Server“-Objekt ermöglicht außerdem eine Suche nach Prozessvariablen innerhalb seines Namensraums.
- dann folgt in der Hierarchie das OPC-Objekt „OPC-Group“ und dient nur zur besseren Strukturierung des nachfolgenden OPC-Objekts.
- die unterste Ebene bildet das OPC-Objekt mit der Bezeichnung „OPC-Item“ und jedes Item besitzt seine eigene ID. Den OPC-Items können Variablen zugeordnet werden, die aber dem Namensraum des OPC-Items entsprechen sollten.

Für die Variablen der Objekt-Items werden vom Server folgende Methoden angeboten:

- Wert auslesen
- Wert ändern
- Wert überwachen: Sobald sich der Wert der vorgegebenen Variablen ändert wird der neue Wert sofort an den Client weitergeleitet.

Auch die Aktualisierungsrate des Zwischenspeichers für die Prozessvariablen am OPC-Server kann vom Client vorgegeben werden.

Bei Leseaufrufen wird zwischen dem Auslesen des Zwischenspeichers am Server und dem direkten Auslesen am Automatisierungsgerät unterschieden. Weiters kann zwischen einem synchronen Lesevorgang (bei dem der Client auf die angeforderten Daten wartet) und einem asynchronen Lesevorgang unterschieden werden. Beim asynchronen Lesevorgang erhält der Client sofort eine Auftragsbestätigung, arbeitet weiter und der neue Variablenwert wird mithilfe einer Ereignismeldung vom Server nachgereicht. Zusätzlich zum Wert der Variable wird immer eine Statusinformation und ein Zeitstempel mitgeliefert.

(vgl. Wellenreuther et al. 2008, S. 732 f.)

(vgl. Autor unbekannt 9. 4. 2009, 1, About OPC – What is OPC?, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC#open)

(vgl. Autor unbekannt 9. 4. 2009, 2, OPC Data Access Specification, http://www.opcfoundation.org/Default.aspx/04_tech/04_spec_da.asp?MID=AboutOPC)

OPC A/E (Alarms & Events)

Zur Übertragung von Alarmen und Ereignissen. Keine kontinuierliche Datenübertragung im Gegensatz zu DA. Die Daten werden nur dann gesendet, wenn ein zuvor konfigurierter Alarm oder Zustand eintritt.

(vgl. Autor unbekannt 9. 4. 2009, 1, About OPC – What is OPC?, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC#open)

OPC Batch

Die bereits in der Einleitung angesprochene Rezeptsteuerung bzw. -verwaltung wird für die Chargenfertigung (batch manufacturing: diskontinuierliches Produktionsverfahren, z.B. mit einem Mischer) verwendet. Diese Spezifikation stellt Schnittstellen und Namensräume für den Austausch von vier Basisdatentypen zwischen Geräten bzw. Systemen unterschiedlicher Hersteller zur Verfügung. Ein Beispiel: Aufgrund einer Änderung eines Rezepts muss ein angepasster Konfigurationsdatensatz vom Prozessleitsystem zu einer Wiegestation übertragen werden, um die Zusammensetzung des Endproduktes verändern zu können.

(vgl. ISA–The Instrumentation, Systems and Automation Society 1995, S. 27 f.)

(vgl. Autor unbekannt 9. 4. 2009, 1, About OPC – What is OPC?, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC#open)

(vgl. Autor unbekannt 9. 4. 2009, 3, OPC Batch Specification, http://www.opcfoundation.org/Default.aspx/04_tech/04_spec_batch.asp?MID=AboutOPC)

OPC DX (Data eXchange)

Diese Spezifikation beschreibt den Datenaustausch zwischen OPC-Servern. Dadurch soll vor allem der Datenaustausch zwischen OPC-Servern unterschiedlicher Hersteller ermöglicht werden ohne einen OPC-Client zwischen die OPC-Server schalten zu müssen. Auch verschiedene Services, wie z.B. ein Diagnose-Service, werden ermöglicht.

(vgl. Autor unbekannt 9. 4. 2009, 1, About OPC – What is OPC?, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC#open)

OPC HDA (Historical Data Access)

Spezifikation zur Übertragung von Protokollen (historisches Archiv). Es werden Schnittstellen beschrieben um Daten von seriellen Log-Geräten bis zu SCADA-Systemen abfragen zu können. Protokollierte Daten dienen unter anderem zur Visualisierung von Diagrammen.

(vgl. Autor unbekannt 9. 4. 2009, 1, About OPC – What is OPC?, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC#open)

OPC Security

Diese Spezifikation beschreibt ein Rechtesystem, um den Zugriff von OPC-Clients auf OPC-Server zu regeln Über OPC-Server werden einerseits teils wertvolle Informationen abgefragt, andererseits können durch unsachgemäße Änderungen von Prozessdaten enorme Verluste durch Produktionsausfälle (Beschädigung von Maschinen, unbrauchbares Produkt, ...) entstehen.

(vgl. Autor unbekannt 9. 4. 2009, 1, About OPC – What is OPC?, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC#open)

OPC XML-DA

OPC XML Data Access basiert auf der Grundlage der XML Web Service Technologie und soll die bisher genutzte COM/DCOM-Technologie ersetzen. Webservices bieten ihre Dienste im Internet an und werden durch ihre URL eindeutig identifiziert. Sie stellen Informationen zur automatischen Weiterverarbeitung zur Verfügung und bieten eine Beschreibung zur Nutzung dieser Dienste an. Diese dienst-orientierte Architektur basiert auf den folgenden Internetprotokollen:

- XML ist eine Auszeichnungssprache. Mithilfe von Tags wird die Information strukturiert und beschrieben. Tags können auch selbst definiert werden.
- WSDL (Web Service Definition Language) bietet eine Beschreibung der Dienste eines Web-Service im XML-Format als Datei an. Diese Datei wird von den OPC XML Clients benötigt.
- SOAP kümmert sich um die Aufrufreihenfolge bei der Kommunikation und das Verpacken der XML Nachrichten. SOAP ist ein Plattform unabhängiges Format und bietet einen „Remote Procedure Call“ Mechanismus.

- HTTP ist ein weit verbreitetes Protokoll zur Übertragung von Nachrichten über das Internet. Aus diesem Grund werden die SOAP Telegramme mithilfe dieses Protokolls über das Internet übertragen.

Webdienste mit der OPC XML-DA Spezifikation besitzen einen ähnlichen Funktionsumfang wie OPC DA. Das Ziel ist die Integration aller bisherigen Schnittstellen in die XML-DA Spezifikation. Vorgesehen sind derzeit folgende Webdienste:

- „GetStatus“ um den Web Service Status in Erfahrung zu bringen
- „Browse“ um den Namensraum durchsuchen zu können
- „Read“
- „Write“
- „Subscribe“ soll als Alternative für die Funktion „automatische Überwachung“ von OPC DA dienen

(vgl. Burbiel 2007, S. 23 f.)

(vgl. Wellenreuther et al. 2008, S. 737 f.)

(vgl. Autor unbekannt 15. 4. 2009, 4, OPC XML-DA Specification, http://www.opcfoundation.org/Default.aspx/04_tech/04_spec_xml.asp?MID=AboutOPC)

OPC UA (Unified Architecture)

Mit OPC UA wird eine neue Architektur eingeführt, die die mit XML-DA begonnene Plattform-Unabhängigkeit beibehalten soll. Die UA-Spezifikation beinhaltet die gesamte Architektur und wurde deshalb in 12 Blöcke aufgeteilt. Die Kompatibilität zu bisherigen OPC-Spezifikationen soll erhalten bleiben und deshalb sind diese in angepasster Form in der UA-Spezifikation enthalten.

(vgl. Autor unbekannt 16. 4. 2009, 5, OPC Unified Architecture, http://www.opcfoundation.org/Default.aspx/01_about/UA.asp?MID=AboutOPC)

2.2 Feldbus

Die Grundidee hinter den Feldbussen war die Dezentralisierung der Peripherie. Bei diesem Konzept werden die Feldgeräte über einen seriellen Bus mit den Automatisierungsgeräten verbunden. Dadurch werden Kosten beim Verdrahtungsaufwand und im Schaltschrankbau eingespart. Ein bekannter Vertreter dieser Art von Feldbus ist zum Beispiel der PROFIBUS DP (Dezentrale Peripherie).

Danach folgte die Dezentralisierung der Steuerungsintelligenz indem sich mehrere Automatisierungsgeräte die Arbeit aufteilten. Dahinter stehen Kosteneinsparungen bei der Softwareentwicklung, Inbetriebnahme und Wartung.

Im dritten Schritt wird das Problem der vertikalen Kommunikation in Angriff genommen. Auch hier ist ein Trend hin zum Ethernet-Bussystem zu erkennen. Ein bekanntes Beispiel dieser Art von Feldbus ist PROFINET. Das bewährte und nicht Ethernet-basierte PROFIBUS-System soll aber nicht verdrängt, sondern integriert werden. Auch hier kommen nur mehr standardisierte und offene Bus-Systeme zum Einsatz.

Um Daten mit einem Automatisierungsgerät über einen Feldbus austauschen zu können, benötigt man einen Treiber, der die Kommunikation mit dem Automatisierungsgerät übernimmt. Nicht jeder Treiber beinhaltet zusätzliche Management-Funktionen zur Verwaltung dieser Geräte. Außerdem sind diese Treiber meist nur für ein Kommunikationsprotokoll zuständig. Treiber für bestimmte Kommunikationsprotokolle werden oft von mehreren Herstellern angeboten. Auch für Linux bzw. Unix sind Treiber erhältlich. Vor allem bei einer Kommunikation mit Automatisierungsgeräten unterschiedlicher Hersteller ist der Einsatz von mehreren Treibern sehr aufwendig.

(vgl. Wellenreuther et al. 2008, S. 505)

2.2.1 Industrial Ethernet

Die Bezeichnung „Industrial Ethernet“ stammt von der Firma Siemens und bezeichnet ein offenes industrielles Bussystem, welches für den Einsatz im Produktionsbereich entwickelt wurde. Industrial Ethernet ist in der Hierarchie oberhalb der nicht Ethernet-basierten Feldbusse angesiedelt und ermöglicht nicht nur die Vernetzung von Automatisierungsgeräten mit einem Leitsystem, sondern bietet auch die Möglichkeit der Anbindung an übergeordnete Firmennetze via Router.

Bei Industrial Ethernet handelt es sich physikalisch um ein normales „Fast Ethernet“. Allerdings sind die Hardwarekomponenten für die raue Umgebung in Produktionsanlagen entwickelt worden. Stecker gibt es zum Beispiel in staub- und wasserdichten Ausführungen. Switches gibt es zum Beispiel mit 24V DC Spannungsversorgung, erweitertem Temperaturbereich und Hutschienen-Montage.

Industrial Ethernet ist kommunikationstechnisch ein Multiprotokollnetz für TCP/IP, ISO-on-TCP (TCP/IP ergänzt um RFC1006 zur Datenübertragung mit variabler Blocklänge) und Simatic S7-Funktionen. Industrial Ethernet ist informationstechnisch ein Netz zur Übertragung großer Datenmengen, aber auch Internetdienste wie HTTP, E-Mail und Ftp werden genutzt. Automatisierungstechnisch ist Industrial Ethernet zur Vernetzung verteilter Automatisierungsgeräte geeignet, welche auch von einem übergeordneten Leitsystem koordiniert werden können. Für die zyklische Prozessdatenübertragung ist die Spezifikation von Industrial Ethernet allein allerdings nicht geeignet.

(vgl. Wellenreuther et al. 2008, S. 565)

2.2.2 PROFINET

Die Kommunikation von PROFINET findet über Industrial Ethernet statt. Es werden folgende Übertragungsarten unterstützt:

- Zyklisch: Zur Übertragung zeitkritischer Daten. Erweiterung zu Industrial Ethernet.
- Azyklisch: Parametrierungs-, Konfigurations- und Diagnosedaten.

Je nach Anforderung werden unterschiedliche Transportprotokolle verwendet:

- TCP/UDP zur Übertragung zeit-unkritischer Daten.
- SRT (Soft Real Time) für zeitkritische PROFINET-Daten bis ca. 10ms Zykluszeit
- IRT (Isochrone Real Time) für harte Echtzeitbedingungen bis 1 ms Zykluszeit. Spezielle Hardware erforderlich.

(vgl. Wellenreuther et al. 2008, S. 584)

PROFINET IO

Dient der Kommunikation mit dezentraler Peripherie bzw. Feldgeräten. Die Kommunikation läuft prinzipiell zyklisch ab. Jedes Feldgerät (Slave) gibt seine Daten an ein Automatisierungsgerät (Master) weiter. Das Konzept und die Konfiguration von PROFINET IO ist ähnlich mit der von PROFIBUS DP. (vgl. Wellenreuther et al. 2008, S. 575)

PROFINET CBA

PROFINET CBA (Component Based Automation) ist ein Konzept für verteilte Automatisierung. Die Kommunikation läuft prinzipiell azyklisch ab. PROFINET CBA kann mit der COM/DCOM-Technologie umgehen und wird deshalb für die Kommunikation zwischen OPC-Server und Automatisierungsgerät eingesetzt. (vgl. Wellenreuther et al. 2008, S. 584)

PROFINET Linux Treiber

Für den PROFINET-Standard werden Treiber für verschiedene Betriebssysteme von unterschiedlichen Herstellern angeboten. Die Auswahl an Linux-Treibern wird allgemein immer größer, da auch die Anzahl der eingesetzten Linux-Systeme kontinuierlich steigt. Diese Treiber können auf üblicher x86-Hardware eingesetzt werden. Mit selbst entwickelten Anwendungen kann zum Beispiel ein Datalogger für bestimmte Prozessdaten erstellt werden. (vgl. Tisken 2008, S. 31)

Auch für Embedded Linux Umgebungen sind bereits Treiber verfügbar. Außerdem wird der Einsatz von Realtime-Linux-Varianten empfohlen. PROFINET Linux Treiber sind für folgende Kanäle möglich:

- TCP-Socket für Standardkanal
- Raw-Socket für „Soft-Real-Time“-Kanal

(vgl. Werner et al. 2005, S. 14 f.)

Für performante Kommunikation über PROFINET bietet die Firma Siemens Kommunikationsprozessoren (CP 16xx) an. Auch für diese PCI-Steckkarten sind Linux-Treiber verfügbar. (vgl. Siemens AG 2007, S. 2)

2.3 Linux Controller von Wago

Im Serverbereich ist die Stabilität von Linux mittlerweile anerkannt. Diese Stabilität ist der Grund warum Linux auch in der Welt der Automatisierungstechnik immer öfter zum Einsatz kommt. In der Medizintechnik ist der Einsatz von Linux auf Embedded-Systemen keine Seltenheit mehr. Die Verbreitung läuft langsam aber stetig ab. Vor allem im Bereich der Software-SPS (Soft-SPS) wird als Betriebssystem oft Linux eingesetzt. Bei dieser Variante erfolgt die Anbindung der I/O-Hardware (Ein- und Ausgänge) über Steckkarten an den PC und die SPS läuft als Programm auf diesem PC. (vgl. Tisken 2008, S. 30 f.)

Mit dem Linux Feldbus-Controller von Wago erhält man ein System, welches Durchgängigkeit vom Controller bis zu den I/Os bietet. Möglich macht das unter anderem der als Kernel-Modul ausgeführte Kbus-Treiber der sich um die Kommunikation zwischen Anwenderprogramm und Hardware kümmert. Mithilfe der Kbus-API wird die Verwendung des Kbus-Treibers in eigenen C-Programmen erleichtert. Diese Durchgängigkeit ist bei Embedded-Systemen sonst nicht zu finden. (vgl. Tisken 2008, S. 33)

Mit dem Linux I/O-IPC (Industrie PC) steht eine wesentlich performantere Alternative neben dem Linux Controller zur Verfügung. Der IPC ist zwar wesentlich breiter als der Controller, dafür bietet der IPC auch mehr Anschlussmöglichkeiten (DVI, USB, RS232, ...), eine schnellere CPU und mehr Speicher, egal ob RAM oder Flash. Das Betriebssystem und die I/O-Module sind im wesentlichen die selben als beim Controller. (vgl. WAGO Kontakttechnik GmbH & Co. KG 2009c, S. 1 f.)

Selbst entwickelte C-Programme für Linux lassen sich ohne viel Aufwand zwischen Controller, IPC und sogar einer Software-SPS von Wago portieren, da bei diesen Varianten die selben I/O-Module verwendet werden können. (vgl. Tisken 2008, S. 33)

2.3.1 I/O-System 750 von Wago

Der Feldbus-Controller ist Teil des WAGO-I/O_SYSTEM 750. An den Feldbus-Controller können Busklemmen für die Ein- und Ausgangsbeschaltung (I/O) mit beliebigen Signalformen angereiht werden. Ein auf diese Weise entstehender Feldbusknoten wird mit einer Endklemme abgeschlossen. Die Kommunikation der Bauteile erfolgt über einen internen Bus, den Klemmenbus (Kbus).

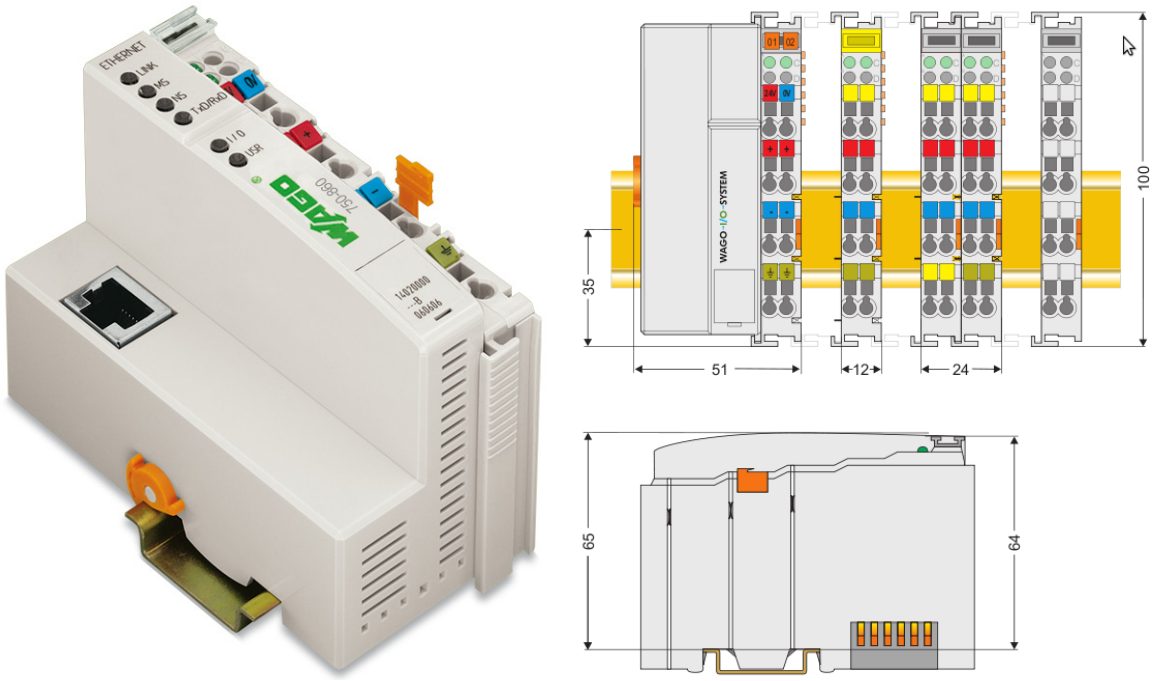

Abbildung 2: Linux Feldbus-Controller (WAGO 2008, S. 1) inkl. Abmessungen (WAGO 2006, S. 19)

Der Feldbus-Controller enthält neben einem Feldbus-Interface eine Elektronik und eine Einspeiseklemme. Das Feldbus-Interface bildet die physikalische Schnittstelle zum Feldbus. Die Elektronik verarbeitet die Daten der Busklemmen und stellt diese für die Feldbus-Kommunikation bereit. Mittels der integrierten Einspeiseklemme werden die 24 Volt der Stromversorgung für das System und die Feldversorgung eingespeist.

Der Feldbus-Controller besteht aus einem ARM-Mikrocontroller und einem uClinux-Betriebssystem mit der Kernelreleaseversion 2.6 (siehe Listing 1). Weiters besitzt er eine Ethernet-Schnittstelle, eine serielle Schnittstelle und ein Klemmenbus-Interface zur Kommunikation mit den I/O-Klemmen.

Der ARM7TDMI ist ein 32-Bit-Prozessor ohne MMU mit RISC-Architektur und einer Taktfrequenz von 44 MHz. Ohne einer MMU erfolgt auch keine Virtualisierung des RAM-Speichers. 16 MB RAM, 32 kB NOVRAM und ein 4 MB großem Flash-Speicher stehen zur Verfügung. Ein 2,5 MB großes JFFS2-Dateisystem ist im Flash-Speicher untergebracht. Auch eine Echtzeituhr (RTC) ist vorhanden (siehe Listing 1).

Der Klemmenbus-Treiber (Kbus-Treiber) ist ein Char-Device-Treiber (/dev/kbus) und als dynamisch ladbares Kernel-Modul ausgeführt. Die Prozessdaten der Busklemmen werden vom Kbus-Treiber in einem Prozessabbild bereitgestellt. Dieses Prozessabbild wird vom Kbus-Treiber im RAM des Feldbus-Controllers abgelegt. Mithilfe der Kbus-API kann von selbst erstellten Anwendungen (C-Programmen) komfortabel auf das Prozessabbild zugegriffen werden.

Ein Automatisierungsgerät arbeitet sein Programm normalerweise zyklisch ab:

1. aktuelle Daten von den Eingangsklemmen in das Prozessabbild schreiben
2. abarbeiten des Anwenderprogramms ausschließlich durch lesen und beschreiben des Prozessabbildes
3. Daten des Prozessabbildes an die Ausgangsklemmen übertragen

Diese Schritte laufen in einer Endlosschleife ab. Damit ein Programmentwickler auf dem Feldbus-Controller diese Funktionalität nachbilden kann, wird ein Abgleich des Prozessabbildes mit den Busklemmen durch den Kbus-Treiber nur auf Befehl ausgeführt. Die Kbus-API stellt dafür den Befehl kbusUpdate() zur Verfügung.

Über die Ethernet-Schnittstelle werden folgende Protokolle unterstützt:

- HTTP; Höchste Anzahl der Socket-Verbindungen: 1
- BootP
- DHCP
- DNS
- SNTP
- FTP; Höchste Anzahl der Socket-Verbindungen: 1
- Telnet; Höchste Anzahl der Socket-Verbindungen: 8
- NFS; Höchste Anzahl der Socket-Verbindungen: 8
- MODBUS/TCP

Im Auslieferungszustand sind folgende Dienste und Anwendungen installiert:

- BOA-Webserver mit CGI-Interface
- FTP-Server
- DHCP-Client
- DNS-Client
- SNTP-Client

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2006, S. 13 ff.)

2.3.2 Linux Systemstart

Nach Einschalten der Spannungsversorgung oder einem Hardware-Reset erfolgt die Initialisierung. Der Feldbus-Controller ermittelt dabei die Anordnung der Busklemmen und erstellt aufgrund dieser Informationen das Prozessabbild.

Beim Hochlauf blinkt die „I/O“-LED (Rot) mit hoher Frequenz. Nach ungefähr 20 Sekunden leuchtet die „I/O“-LED grün auf. Dann ist der Feldbus-Controller betriebsbereit und der Klemmenbus-Treiber wurde geladen. Tritt während des Hochlaufens ein Fehler auf, so wird dieser über die „I/O“-LED durch Blinken in roter Farbe anhand folgender Blinkcodes angezeigt:

- Nach der ersten Blinksequenz (ca. 10 Hz) beginnt die Fehleranzeige.
- Nach einer Pause folgt die zweite Blinksequenz (ca. 1 Hz) und die Anzahl der Blinkzeichen gibt den Fehlercode an.
- Die dritte Blinksequenz (ca. 1 Hz) erfolgt nach einer weiteren Pause und wieder gibt die Anzahl der Blinkzeichen das Fehlerargument an.

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2006, S. 54 f.)

```
Linux version 2.6.14-hsc0 (u01896@pc92960) (gcc version 3.4.3) #117
PREEMPT Tue Feb 27 11:56:17 CET 2007
CPU: ARM7TDMI [41007700]
Machine: Digi/NetSilicon NET+ARM
Memory management: Non-Paged(unused/noMMU)
...
Memory: 15MB = 15MB total
Memory: 14156KB available (1560K code, 153K data, 68K init)
...
Creating 5 MTD partitions on "WAGO FBK 750 w. U-Boot":
0x00000000-0x00010000 : "RESERVED AREA"
0x00010000-0x00040000 : "Bootloader"
0x00040000-0x002d0000 : "Filesystem"
0x002d0000-0x003f0000 : "Linux Kernel"
0x003f0000-0x00400000 : "BootUp Parameter"
...
RTC8564 driver version 1.0.0a (03/31/06)
drivers/i2c/busses/i2c-na_ppio.c: rtc8564 registered.
...
VFS: Mounted root (jffs2 filesystem).
...
kbus: module license 'Proprietary. Send bug reports to
support@wago.com' taints kernel.
...
Klemme 750-4xx entdeckt.
Klemme 750-4xx entdeckt.
Klemme 750-5xx entdeckt.
Klemme 750-459/000-000 entdeckt.
Klemme 750-559/000-000 entdeckt.
Klemme 750-5xx entdeckt.
Klemme 750-650/000-000 entdeckt.
...
```

Listing 1: Terminalausgabe beim Systemstart des Testcontrollers: Kernelversion, CPU, RAM, Partitionen, Echtzeituhr, Journaling-Flash-FileSystem, Kbus-Treiber, angeschlossene Busklemmen

2.3.3 Verbindungsaufbau mit der Konsole des Feldbus-Controllers

Am einfachsten gelingt der Verbindungsaufbau mit dem Feldbus-Controller mithilfe eines Telnet-Clients. Ein Verbindungsaufbau ist auch über die serielle Schnittstelle möglich, allerdings ist diese Variante aufwendiger und es wird ein spezielles Verbindungskabel benötigt. Falls die IP-Adresse verstellt worden ist und nicht mehr in Erfahrung gebracht werden kann, gestaltet sich der Verbindungsaufbau via serieller Schnittstelle einfacher.

```
chris@chrux:~$ telnet 192.168.1.213
Trying 192.168.1.213...
Connected to 192.168.1.213.
Escape character is '^]'.

NET+ARM login: root
Password:

    Welcome to

 *------------------------*
 | 750-860 WAGO-Linux-FBC |
 | Version: 2.1.3 (3)     |
 *------------------------*

For further information check:
http://www.wago.com/



BusyBox v1.00 (2006.12.12-15:36+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.


~ #
~ # help

Built-in commands:
-------------------
      . : break cd continue eval exec exit export help login newgrp
      read readonly set shift times trap umask wait [ busybox cat chgrp
      chmod chown chroot clear cp date df dmesg du echo env false free
      ftpget ftpput grep hostname hwclock ifconfig insmod kill killall
      klogd ln ls lsmod mkdir mknod modprobe more mount msh mv ping
      pivot_root ps pwd reboot reset rm rmdir rmmod route sh sleep
      syslogd tail telnet test tftp time top touch true tty umount
      uptime vi which
```

Listing 2: Telnet-Login am Testcontroller mit Ausgabe der durch die Busybox zur Verfügung gestellten Befehle bzw. Programme

Die Ausgabemöglichkeit der zur Verfügung stehenden Shell-Befehle ist insofern ein guter Anhaltspunkt, als das der Befehlsumfang im Vergleich zu einer Linux Version auf einem x86-System nicht dem selben Umfang entspricht.

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2006, S. 51 ff.)

2.3.4 Das Dateisystem des Feldbus-Controllers

Es soll hier nur kurz auf die wichtigsten Verzeichnisse am Feldbus-Controller eingegangen werden.

Das /proc Verzeichnis

Der Linux-Kernel legt im /proc-Verzeichnis dynamisch (dieses Verzeichnis existiert nur im RAM) Informationen zu jedem laufenden Prozess und dem Kernel selbst ab. Auch Informationen zum geladenen Kbus-Treiber, welcher das Prozessabbild der Ein- und Ausgänge den User-Programmen zur Verfügung stellt, sind abrufbar.

```
/proc # cat driver/kbus/status
Command Errors:   0     max.: 2000
Timeout Errors:   0     max.: 200
Input Data Errors:      0     max.: 6500
Output Data Errors:     0     max.: 6500
/proc # cat driver/kbus/config
Slot  Terminal    Complex     OutByteOffset     OutBitOffset
      InByteOffset      InBitOffset
1     750-4xx           n     x           x           12          0
2     750-4xx           n     x           x           12          2
3     750-5xx           n     12          0           x           x
4     750-459/000-000   y     x           x           0           0
5     750-559/000-000   y     0           0           x           x
6     750-5xx           n     12          4           x           x
7     750-650/000-000   y     8           0           8           0
PAB In:     00ffe060     PAB Size: 2040 bytes
PAB Out:    00ffe858     PAB Size: 2040 bytes
```

Listing 3: Informationen zum Kbus-Treiber im /proc-Verzeichnis: aktuell am Feldbus-Controller angeschlossene Busklemmen inkl. Offsets im Eingangs- und Ausgangsprozessabbild

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2006, S. 63)

Das /etc Verzeichnis

Im /etc-Verzeichnis werden die Konfigurationsdateien für diverse Programme und Dienste abgelegt. Die Datei „rc“ wird beim Systemstart von Linux standardmäßig aufgerufen.

```
/etc # cat rc
#!/bin/msh
hostname NET+ARM
mount -t proc none /proc
mount -t devpts none /dev/pts
/bin/inetd &
/etc/startwago
```

Listing 4: Beim Systemstart wird die Datei "rc" im /etc-Verzeichnis aufgerufen. Diese ruft unter anderem auch die Feldbus-Controller-spezifische Datei "startwago" auf.

Diese wiederum ruft die Datei „startwago“ auf, welche dann Dienste und Treiber startet, die nur für den Feldbus-Controller relevant sind.

```
/etc # cat startwago
#!/bin/msh

# change setuid/setgid-Bit for tinylogin(su)
chmod +s /bin/tinylogin

# start the KBus-Driver
insmod kbus

# start modbus tcp server
/bin/mb_tcp &

# start web server (BOA)
boa -c /etc/boa &

# if it is configured, start the IOCheck-Driver/-Application
if grep -q "wago_console=yes" /proc/cmdline
then
  rm /etc/inittab
  ln -s /etc/wago_console/inittab.nottyS0 /etc/inittab
  # start the driver for reading BK-Table
  insmod iocheck
else
  rm /etc/inittab
  ln -s /etc/wago_console/inittab.ttyS0 /etc/inittab
  insmod iocheck
fi

# MEINS
#/bin/msntp -r -P no -x 9 at.pool.ntp.org &
/bin/date
/bin/msntp -r -P no at.pool.ntp.org
/bin/msntp -r -P no at.pool.ntp.org
/bin/msntp -r -P no at.pool.ntp.org
/bin/date

exit 0
/etc #
```

Listing 5: In der Datei "startwago" werden wichtige Dienste für den Feldbus-Controller gestartet: Kbus-Treiber, Modbus TCP (ein Feldbus), BOA Webserver und Terminal für die serielle Schnittstelle in Abhängigkeit des Schiebeschalters

Bei meinem Testcontroller verstellte sich die Systemzeit nach jedem Spannungsausfall auf das Jahr 1970. Im Handbuch des Feldbus-Controllers wird allerdings eine Betriebsdauer der Hardwareuhr (RTC) ohne Spannungsversorgung von mindestens 6 Tagen angegeben (vgl. WAGO 2006, S. 58). Mit dem „msntp“-Befehl können die Systemzeit und die Hardwareuhr anhand eines Zeitservers eingestellt werden. Es liegt also der Verdacht nahe, dass das sporadische Auftreten von Fehlermeldungen beim Ausführen des „msntp“-Befehls mit einem Defekt der Hardwareuhr zusammenhängt.

```
/etc # ps |grep bus
   22 root            SW< [KbusThread]
```

Listing 6: Nach dem Systemstart des Testcontrollers läuft der Kbus-Thread bzw. Kbus-Treiber im Hintergrund, welcher durch die Datei "startwago" aufgerufen wird.

Das /lib/modules Verzeichnis

Die von der Firma Wago zur Verfügung gestellten Kernel-Module befinden sich allerdings in dem Verzeichnis /lib/modules/2.6.14-hsc0/kernel. Diese Kernel-Module sind für den Betrieb des Feldbus-Controllers als Automatisierungsgerät unerlässlich.

```
/lib/modules # ls -alih 2.6.14-hsc0/kernel/
     12 drwxr-xr-x    4 root     root            0 Mar  9  2007 .
     11 drwxr-xr-x    3 root     root            0 Mar  9  2007 ..
     14 drwxr-xr-x    2 root     root            0 Mar  9  2007 IOCheck
     13 drwxr-xr-x    2 root     root            0 Mar  9  2007 Kbus
```

Listing 7: Die am Testcontroller verfügbaren Wago-Treiber

Das /var/www/cgi-bin Verzeichnis

In diesem Ordner werden die CGI-Programme für den BOA-Webserver abgelegt. Hier befinden sich viele Links auf das eigentliche Programm „web“. Dadurch wird kostbarer Speicherplatz auf dem Feldbus-Controller gespart. Diese Dateien werden beim Aufruf des Wago-Web-based Managment vom Boa-Webserver ausgeführt.

```
/var/www/cgi-bin # ls -alih
  26 drwxrwxrwx    2 root     root        0 Apr  1  2009 .
  22 drwxrwxrwx    5 root     root        0 Jan  8  2007 ..
 228 lrwxrwxrwx    1 root     root        3 Mar  9  2007 config -> web
 229 lrwxrwxrwx    1 root     root        3 Mar  9  2007 eeprom -> web
 227 lrwxrwxrwx    1 root     root        3 Mar  9  2007 ether -> web
 225 lrwxrwxrwx    1 root     root        3 Mar  9  2007 info -> web
 230 lrwxrwxrwx    1 root     root        3 Mar  9  2007 kernelmsg ->
web
 226 lrwxrwxrwx    1 root     root        3 Mar  9  2007 restart -> web
 438 lrwxrwxrwx    1 root     root        6 Apr  1  2009 sd -> senddo
 437 -rwxrwxrwx    1 root     root    15.6k Apr  1  2009 senddo
 224 -rwxr-xr-x    1 root     root    89.0k Mar  8  2007 web
```

Listing 8: Das CGI-Verzeichnis für den BOA-Webserver mit dem CGI-Programm inkl. Links für das Wago-Web-based Management

2.3.5 Web-based Management des Feldbus-Controllers

Diese Webseiten sind werksseitig vorinstalliert. Es können nicht nur allgemeine Informationen über den Feldbus-Controller abgerufen werden, sondern auch grundlegende Änderungen durchgeführt werden. Dafür ist allerdings eine Authentifizierung notwendig. Zusätzlich können auch die I/O-Klemmen am Klemmenbus kontrolliert werden und sogar ein Reboot des Systems ist über die Management-Webseite möglich.

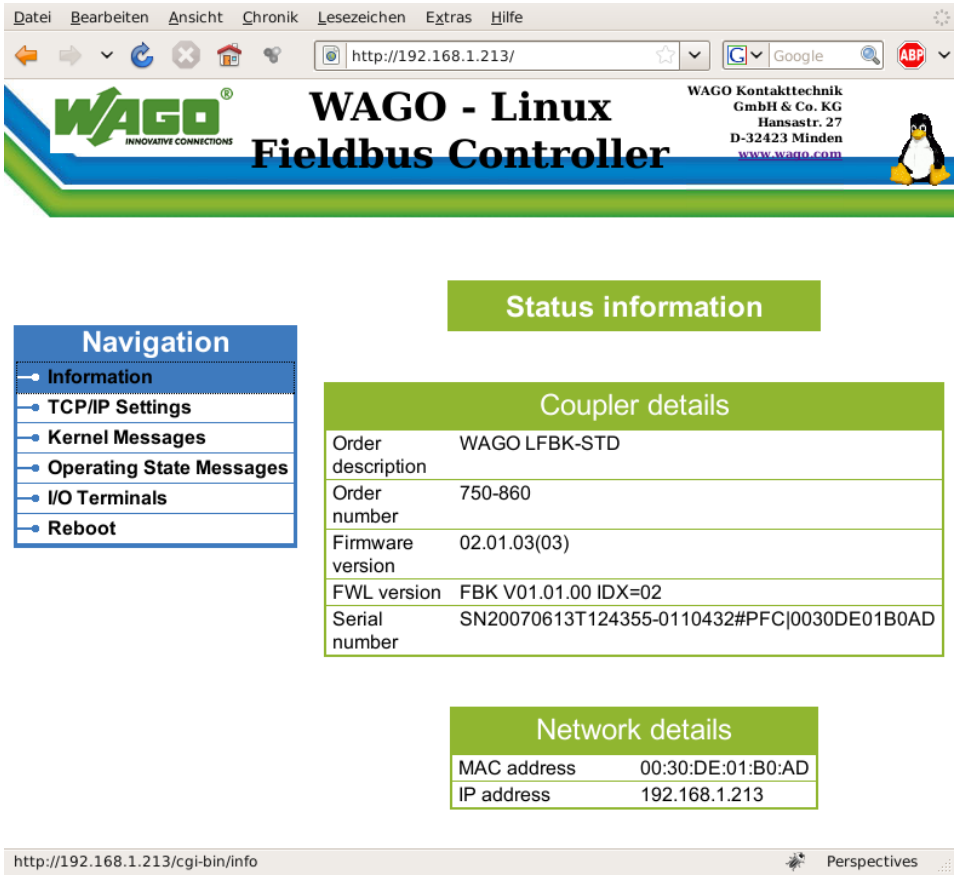

Abbildung 3: Web-based Management: Information

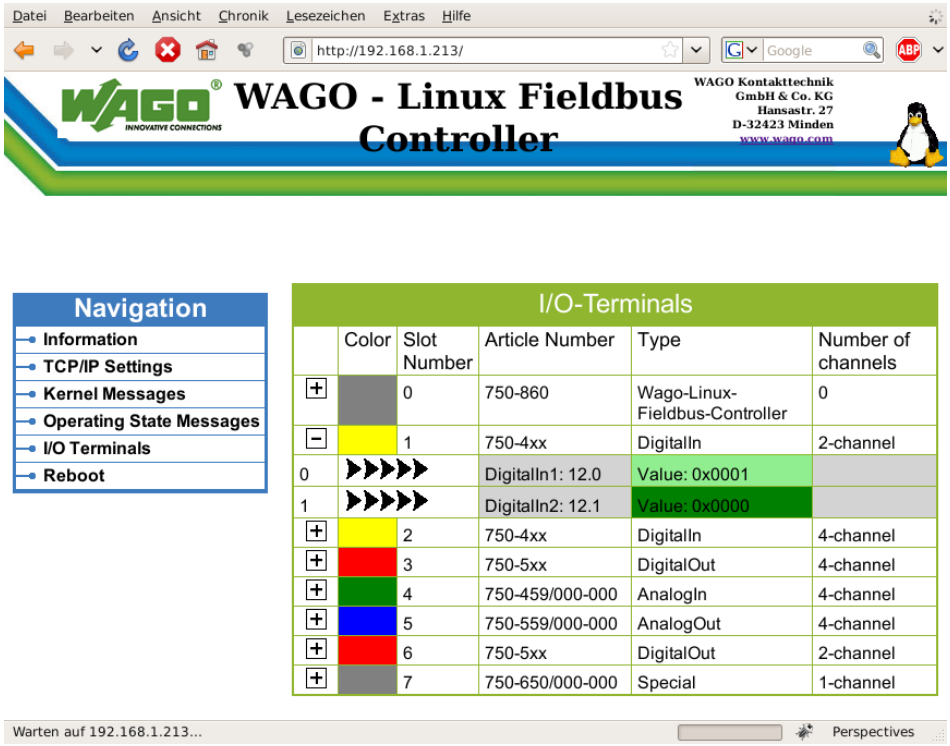

Abbildung 4: Web-based Management: I/O-Busklemmen

2.3.6 Prozessabbild am Feldbus-Controller

Beim Systemstart lädt der Feldbus-Controller automatisch den Klemmenbus-Treiber. Dieser erkennt alle am Klemmenbus gesteckten Busklemmen (Ein- und Ausgänge, siehe Abbildung 5), die Daten liefern bzw. erwarten. Aus der Datenbreite und dem Typ der Busklemme sowie der Position der Busklemmen erstellt der Feldbus-Controller ein internes lokales Prozessabbild, welches in einen Eingangs- und Ausgangsdatenbereich unterteilt ist. Für das Ein- und Ausgangsprozessabbild werden die Daten der Busklemmen in der Reihenfolge ihrer Position am Feldbus-Controller in dem jeweiligen Prozessabbild abgelegt. Zuerst werden die byte-orientierten und dann im Anschluss die bit-orientierten Busklemmen in das Prozessabbild übernommen. Die Bits digitaler Klemmen werden zu Bytes erweitert und wenn die Anzahl der digitalen I/Os größer als 8 Bit ist, dann wird automatisch ein weiteres Byte angelegt.

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2006, S. 131 f.)

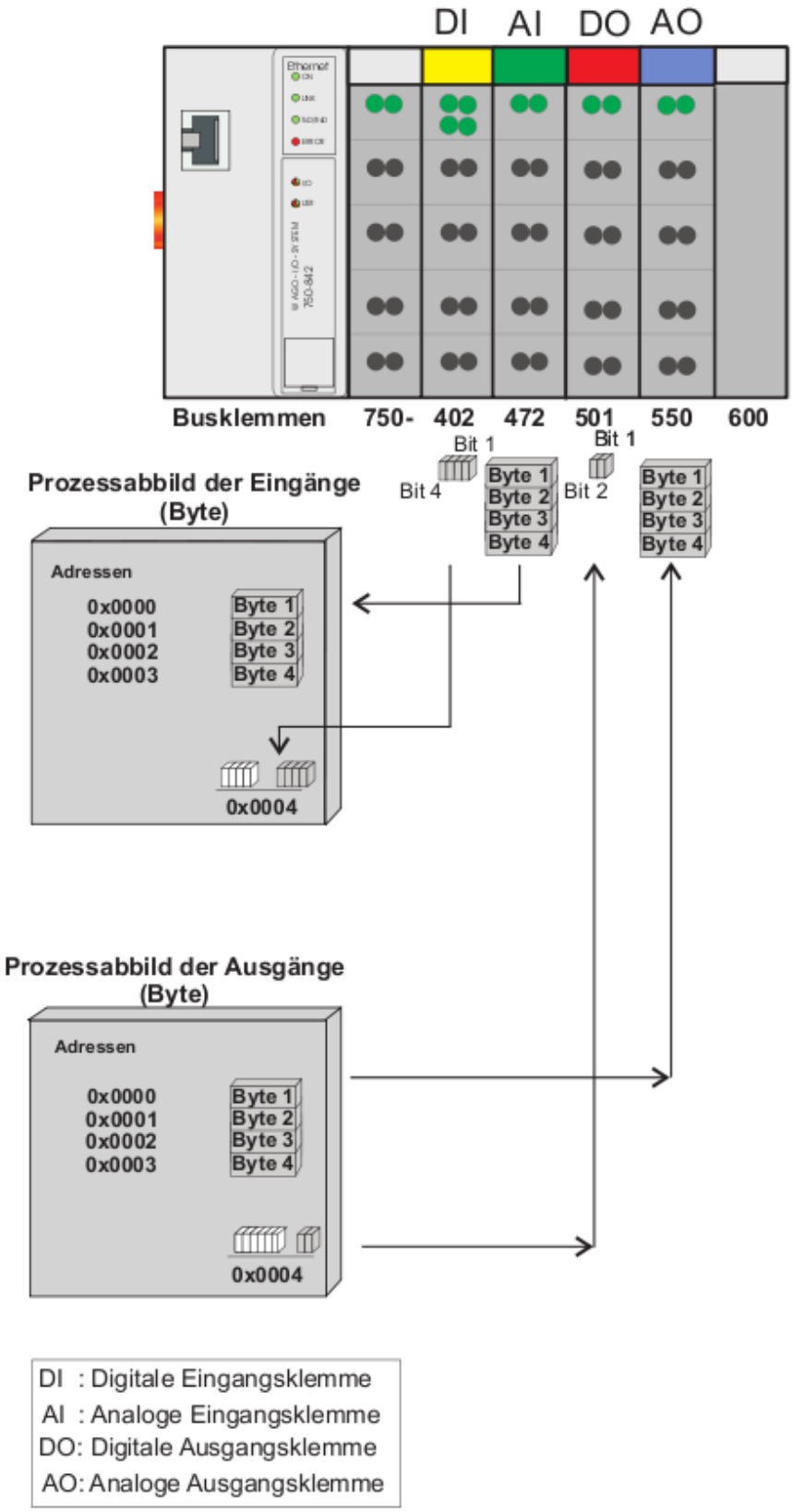

Abbildung 5: Beispiel der Adressierung im Prozessabbild des Feldbus-Controllers (WAGO 2006, S. 1)

Kbus-API

Wie bereits erwähnt, ermöglicht die Kbus-API einen komfortablen Zugriff auf den Kbus-Treiber. Die Header Datei (kbusapi.h) muss dafür in das eigene C-Programm eingebunden werden (#include "kbusapi.h").

```
#include <asm/types.h>

#define IOCTL_KBUSUPDATE           1
#define IOCTL_GETBININPUTOFFSET    2
#define IOCTL_GETBINOUTPUTOFFSET   3
#define IOCTL_CHANGE_UPDATE_SPEED  4

typedef enum { FALSE, TRUE } BOOL;

#define PAB_SIZE 1020
typedef union
{
     struct   /* Zugriff auf PAB als u16 */
     {
       __u16  Pab [PAB_SIZE];       /* Prozess-Abbild */

       #if PFC
         __u16  Var [PFC_VAR_SIZE]; /* Variablen-Bereich, direkt hinter
PAB */
       #endif
     } us;

     struct   /* Zugriff auf PAB als u8 */
     {
       __u8 Pab [sizeof (__u16)*PAB_SIZE];       /* Prozess-Abbild */

       #if PFC
         __u8 Var [sizeof (__u16)*PFC_VAR_SIZE]; /* Variablen-Bereich,
direkt hinter PAB */
       #endif
     } uc;

}T_PabVarUnion;

#define PABIN (volatile T_PabVarUnion *) 0xFFE060;
#define PABOUT (volatile T_PabVarUnion *)
(0xFFE060+sizeof(T_PabVarUnion));
static volatile T_PabVarUnion * pstPabOUT = PABOUT;
static volatile T_PabVarUnion * pstPabIN = PABIN;

extern int KbusOpen(void);
extern int KbusUpdate(void);
extern int KbusClose(void);
extern int KbusGetBinaryInputOffset(void);
extern int KbusGetBinaryOutputOffset(void);
```

Listing 9: kbusapi.h: Die Header Datei der Kbus-API

Anhand der Datei kbusapi.c ist der Umgang mit dem Kbus-Treiber ersichtlich. Die Kbus-API muss zwar nicht für eigene Programme verwendet werden, unter anderem trägt sie aber auch zur besseren Übersicht in Programmen bei.

```
...
#include "kbusapi.h"
int iFD;

static void modifyPABforEffect(void) {
  pstPabOUT->us.Pab[0] = pstPabOUT->us.Pab[0] << 1;
  if(pstPabOUT->us.Pab[0] > 0xFFF) pstPabOUT->us.Pab[0] = 1;
  return;
}

// ------------------------------------------------------------------
/// Oeffnet einen Kanal zum Kbus-Treiber und initialisert die
Bibliothek.
/// \retval 0 Kbus-Schnittstelle bereit
/// \retval EACCES Kein Zugriff auf /dev/kbus moeglich
/// \retval ENODEV /dev/kbus existiert nicht auf dem Dateisystem
/// \retval ENOMEM nicht genug Speicher verfuegbar
/// \retval EMFILE zuviele Dateien auf dem System bereits geoeffnet
/// \retval EIO low-level IO Fehler
// ------------------------------------------------------------------
int KbusOpen() {
  iFD = open("/dev/kbus", O_WRONLY);

  if(iFD < 1) {
    printf("KBUSAPI: Failed opening fifo for writing: %s",
strerror(errno));
    return errno;
  }
  return 0;
}

// ------------------------------------------------------------------
/// Aktualisiert Prozessdaten und Prozessabbild. Kann nur nach
KbusOpen()
/// aufgerufen werden.
/// \retval 0 Aktualisierungsnachricht erfolgreich gesendet
/// \retval EINVAL Kanal zum Kbus wurde nicht oder fehlerhaft geoeffnet
// ------------------------------------------------------------------
int KbusUpdate() {
  int iBytes=0;
  int iTmp;

  if(iFD < 1) return -EINVAL;

  iBytes = ioctl(iFD, IOCTL_KBUSUPDATE, &iTmp);
  if (0 >= iBytes)
    return -EINVAL;
  return 0;
}
```

Listing 10: kbusapi.c: Der Quellcode der Kbus-API, 1. Teil

```
// ------------------------------------------------------------------
// ------------------------------------------------------------------
int KbusGetBinaryInputOffset() {
  int iBytes = 0;
  int iInputOffset = 0;

  if(iFD < 1) return -EINVAL;

  iBytes = ioctl(iFD, IOCTL_GETBININPUTOFFSET, &iInputOffset);
  if (0 >= iBytes)
    return -EINVAL;
  return iInputOffset;
}

// ------------------------------------------------------------------
// ------------------------------------------------------------------
int KbusGetBinaryOutputOffset() {
  int iBytes = 0;
  int iOutputOffset = 0;

  if(iFD < 1) return -EINVAL;

  iBytes = ioctl(iFD, IOCTL_GETBINOUTPUTOFFSET, &iOutputOffset);
  if (0 >= iBytes)
    return -EINVAL;
  return iOutputOffset;
}

// ------------------------------------------------------------------
/// Schliesst den Kanal zum Kbus und gibt allozierte Resourcen wieder
frei.
/// \retval 0 Kanal geschlossen
/// \retval EINVAL Kbus-Kanal war nicht (erfolgreich) geoeffnet
// ------------------------------------------------------------------
int KbusClose() {
  /* Close /dev/kbus */
  close(iFD);
  return 0;
}
```

Listing 11: kbusapi.c: Der Quellcode der Kbus-API, 2. Teil

2.3.7 Beispielprogramme am Feldbus-Controller

Am Feldbus-Controller sind einige Beispielprogramme zu Demonstrationszwecken vorinstalliert. Diese Beispiele zeigen sehr gut den Umgang mit dem Feldbus-Controller auf und sind ein guter Ausgangspunkt um eigene C-Programme für den Feldbus-Controller zu erstellen.

Programm zum Lesen der Eingänge und setzen von Ausgängen

Dieses Programm verwendet die „kbusapi“, welche den Zugriff auf den kKbus-Treiber wesentlich vereinfacht. Eingangsdaten werden Byte-weise ausgelesen, die Ausgänge jedoch Bit-weise gesetzt. Dieses Beispielprogramm trägt wesentlich zum Verständnis der Kbus-API und dessen Umgang mit dem Prozessabbild und des Kbus-Treibers bei, weshalb dieses Programm auch als Vorlage für das C-Programm am Feldbus-Controller im nachfolgenden Projekt diente.

```
/usr/bin # kbusdemo --help
Usage: kbusdemo OPTION [ADDRESS] [OPTION] [ADDRESS] ...

WAGO KBUS DEMO.

OPTION:
  -r, --read    Read Inputbyte [ADDRESS=xxx]
  -n, --on      Switch on Outputbit [ADDRESS=xxx.x]
  -o, --off     Switch off Outputbit [ADDRESS=xxx.x]
  -h, --help    Print this message.

ADDRESS:
  xxx.x         Startaddress (Byte.Bit) from Output/Input

Proprietary. Send bug reports to support@wago.com
```

Listing 12: Mit dem Beispiel-Programm "kbusdemo" können nicht nur die Eingänge der Busklemmen Byte-weise ausgelesen werden, sondern auch die Ausgänge der Busklemmen Bit-weise gesetzt werden.

```
/usr/bin # kbusdemo -r 12
IN[12]: 0b00100101  ||  IN[12]: 0x25
```

Listing 13: Auslesen des 13. Bytes aus dem Eingangs-Prozessabbild. Zuerst werden die einzelnen Bits dargestellt, dann als Hexadezimalzahl.

```
/usr/bin # kbusdemo -n 12.1
OUT[12]: 0 --> OUT[12]: 2

/usr/bin # kbusdemo -n 12.0
OUT[12]: 2 --> OUT[12]: 3
```

Listing 14: Setzen der Bits 12.1 und 12.0 im Ausgangs-Prozessabbild. In der Ausgabe erscheint das entsprechende Byte als Integerzahl bevor und nachdem das Prozessabbild geändert wurde.

Wie bereits erwähnt ist der Kbus-Treiber für den Abgleich des Prozessabbildes mit den Busklemmen zuständig. Dabei werden die Daten von den Eingangs-Busklemmen in das Eingangs-Prozessabbild übertragen. Die Daten des Ausgangs-Prozessabbildes werden an die jeweiligen Ausgangs-Busklemmen übergeben.

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2006, S. 171 f.)

3 Projekt

3.1 Aufgabenstellung

Als Aufgabenstellung wurde ein Summenstörmeldesystem ausgewählt. Die Aufgabe eines solchen Systems besteht in der Visualisierung vieler einzelner Störmeldungen von unabhängigen Teilsystemen in einem Industriebetrieb. In so einem Betrieb arbeiten viele Maschinen oft vollkommen autonom. Tritt eine Störung auf, würde das dem Personal nicht sofort auffallen, besonders in der Nacht oder am Wochenende. Aus diesem Grund werden solche Maschinen bzw. Anlagen an ein Summenstörmeldesystem angeschlossen. Dadurch werden sämtliche Störmeldungen zusammengefasst und können von einer Position aus komplett überwacht werden.

Die ersten Summenstörmeldesysteme wurden zur Signalisierung nur mit Lampen und/oder Hupen ausgerüstet. Nähere Details zur Störung waren somit nicht vorhanden. Diese sind von Systemen mit SPS und Bedienterminal abgelöst worden. Allerdings sind auch diese Bedienterminals ortsgebunden. Mehrere Bedienterminals sind möglich, aber diese sind auch ortsgebunden und die Kosten pro Bedienterminal sind nicht unerheblich.

Eine Vorgabe für meine Projektumsetzung besteht darin, als Ausgabeterminal einen Webbrowser einzusetzen. PC's mit Webbrowser sind heutzutage überall verfügbar, eine Installation von speziellen Programmen entfällt. Somit kann der aktuelle Status der Störmeldung überall abgerufen werden, was je nach Konfiguration auch außerhalb des Werksgeländes möglich ist.

Eine weitere Vorgabe setzt die Verwendung einer Datenbank voraus. Darin sollen alle Störmeldungen zusammen gefasst werden und können somit zentral verwaltet werden. Auch eine Logging Funktion ist somit einfach nachrüstbar. Weiters kümmert sich die Datenbank auch um eventuell auftretende gleichzeitige Datenzugriffe.

Auch eine Quittierungsfunktion soll realisiert werden. Dabei werden aktuelle aktive Störmeldungen akzeptiert, also quittiert. Deren Status ändert sich dadurch und somit können neu auftretende aktive Störmeldungen vom Anwender leichter wahrgenommen werden. Die Gefahr eines Übersehen von neuen aktiven Störmeldungen soll so weitgehend eliminiert werden.

3.2 Gesamtkonzept

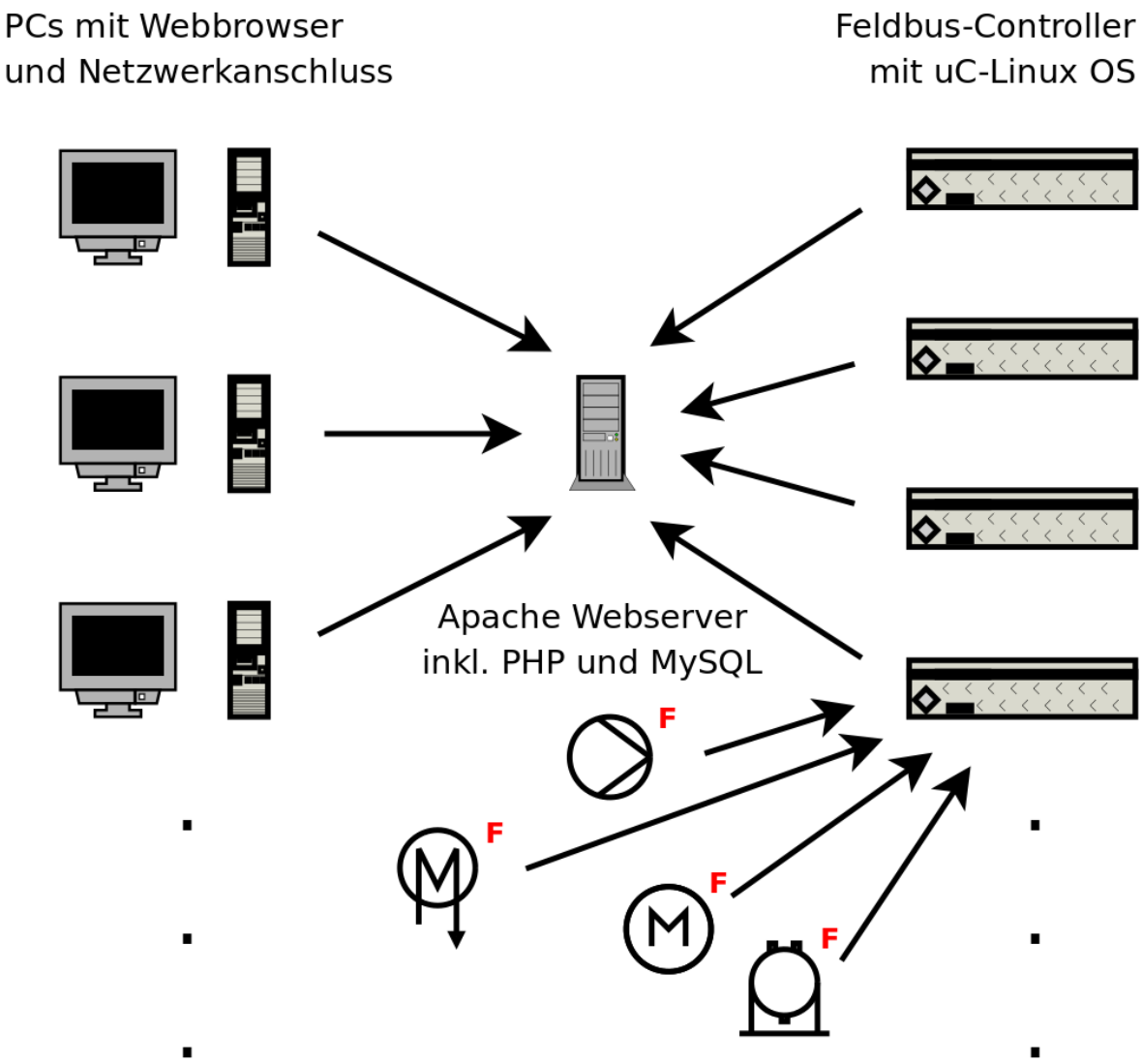

Abbildung 6: Gesamtkonzept für ein Summenstörmeldesystem
(F: Fehler- bzw. Störmeldung)

Das zentrale Element des Summenstörmeldesystems bildet der Webserver, der die aktuellen Daten in einer Datenbank verwaltet. Tritt eine Störung auf, werden die aktualisierten Daten von dem jeweiligen Feldbus-Controller an den Webserver gesendet und in der Datenbank abgelegt. Anfragen von Webbrowsern nach der Summenstörmeldeliste werden vom Webserver beantwortet und mit Hilfe des in Kapitel 3.2.1 näher beschriebenen „erweiterten Periodic Refresh“ Pattern Client-seitig aktuell gehalten.

3.2.1 Webbrowser

Zum Abrufen der Summenstörmeldeliste wird ein Webbrowser mit aktiviertem Javascript benötigt. Die Liste wird mit Ajax aktuell gehalten, indem eine Anfrage nach Veränderung der Daten an den Webserver gesendet wird. Eine Antwort wird vom Webserver erst zurück geschickt, wenn sich die Daten in der Datenbank geändert haben. Um keine Timeouts im Webbrowser noch am Webserver zu erhalten, wird nach einer festgelegten Zeitspanne auch ohne einer Änderung der Daten eine Antwort vom Webserver an den Webbrowser geschickt. Nachdem die Antwort am Webbrowser eingetroffen ist, wird erneut eine Anfrage an den Webserver gesendet. Es handelt sich hierbei also um ein erweitertes „Periodic Refresh“ Design Pattern. (vgl. Mahemoff 2006, S. 215 f.)

Timeouts lassen sich normalerweise verändern bzw. verlängern, wenn nicht sogar deaktivieren. Während diese Änderung am Webserver über längere Zeit problemlos funktionieren kann, so kann die Änderung der Timeouts im Webbrowser relativ rasch Probleme verursachen. Nutzt der Webserver nämlich die verlängerten Timeouts aus, führt dies auf nicht angepassten Webbrowsern erst recht zu einem Timeout. Jede Änderung von Timeouts sollte auf alle Fälle gut bedacht werden.

3.2.2 Webserver

Der Webserver ist für folgende Anfragen der Webbrowser zuständig:

- Anfrage nach der Webseite: Die Datei „index.html“ wird an den Webbrowser gesendet. Diese beinhaltet auch Javascript für Client-seitige Funktionalität.
- Anfrage nach der Störmeldeliste: Die aktuellen Störmeldungen aus der Datenbank werden in eine Tabelle eingefügt und an den Webbrowser gesendet. Dieser aktualisiert die Ansicht mit der neuen Störmeldeliste.
- Anfrage mit Zeitstempel auf Veränderung: Sobald der Zeitstempel des Webbrowsers nicht mehr mit dem der Datenbank übereinstimmt, wird dies dem Webbrowser sofort mitgeteilt. Wie bereits erwähnt wird nach einer festgelegten Zeitspanne jedenfalls eine Antwort gesendet.
- Anfrage nach Quittierung: Aktive Störmeldungen werden in der Datenbank auf „quittiert“ gesetzt und eine Antwort an den Webbrowser gesendet.

Der Webserver nimmt auch die Daten vom Feldbus-Controller entgegen und legt diese in der Datenbank ab. Danach wird der Zeitstempel in der Datenbank aktualisiert. Wartende Anfragen auf Veränderung von den Webbrowsern werden in Folge mit einer positiven Antwort zurück geschickt.

3.2.3 Feldbus-Controller

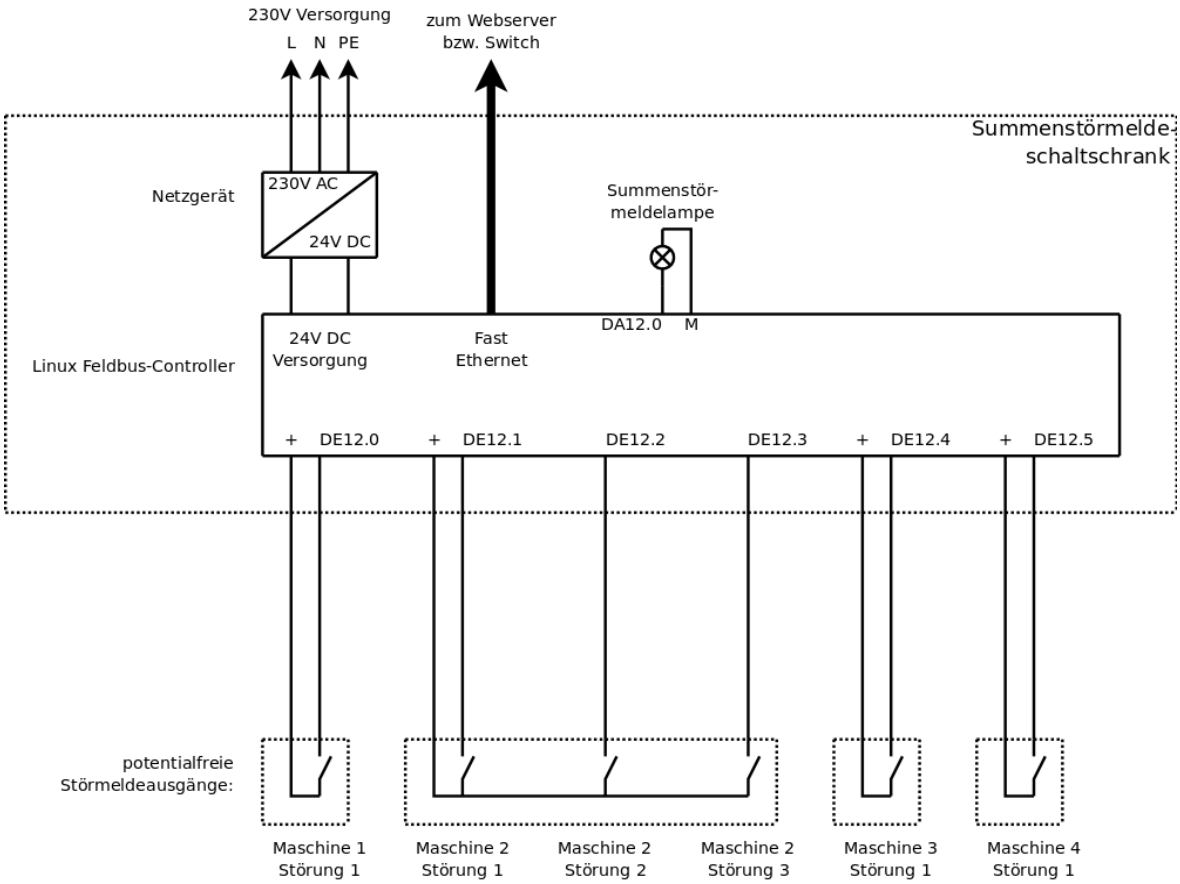

Abbildung 7: Beispielbeschaltung des Feldbus-Controllers für ein Summenstörmeldesystem

Am Feldbus-Controller sind Busklemmen mit digitalen Eingängen montiert bzw. angesteckt. An die digitalen Eingänge werden potentialfreie Kontakte (Relaiskontakt, Endschalter, ...) von Geräten oder Maschinen, welche eine Störung signalisieren, angeschlossen. Solche Störmeldungen können zum Beispiel sein:

- Hilfsschalter eines Leitungsschutzschalters in einem Zählerkasten.
- Hilfsschalter eines Motorschutzschalters (Pumpe, Ventilator, …).

- Thermostat in einem Serverraum.
- Überlaufmeldung eines Abwassertanks.
- Störmelderelais einer Klimaanlage
- Gassensor in einem Lagerraum

Diese digitalen Eingänge (Störmeldeeingänge) werden kontinuierlich vom Feldbus-Controller abgefragt. Ändert sich der Zustand eines Eingangs, dann werden die Daten mittels eines HTTP-Requests als JSON-Objekt an den Webserver gesendet und in dessen Datenbank abgelegt.

Ursprünglich war der Einsatz eines MySQL-Clients am Feldbus-Controller vorgesehen um die Daten ohne Umweg über den Webserver direkt in der Datenbank ablegen zu können. Das Programm ließ sich jedoch nicht für den Feldbus-Controller kompilieren.

3.3 Testaufbau

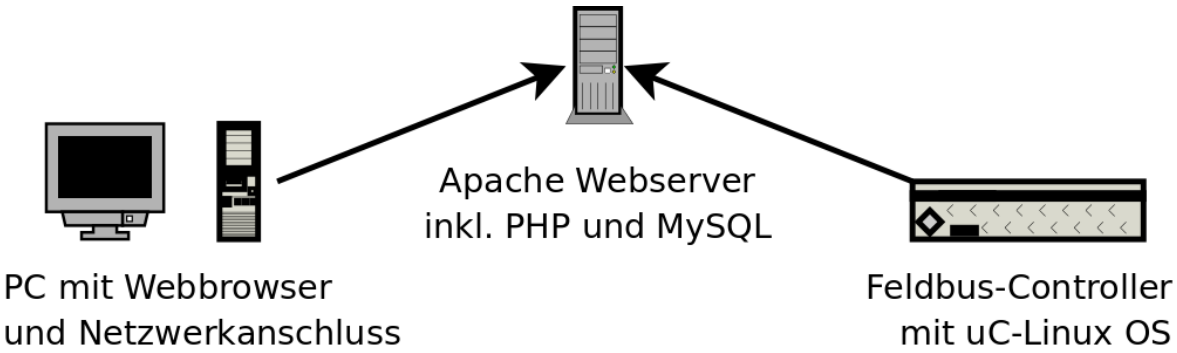

Abbildung 8: Übersicht des Testaufbaus für das Summenstörmeldesystem

Im Gegensatz zum Gesamtkonzept wurde beim Testaufbau nur jeweils ein Gerät verwendet:

- Feldbus-Controller
- Test-PC mit Webbrowser
- Webserver mit Datenbank

Der eingesetzte Webserver ist öffentlich zugänglich und die Störmeldeliste konnte auch von der Fachhochschule aus betrachtet werden. Außerdem sind die Skripte und

Programme nicht für den Produktiveinsatz optimiert.

3.3.1 Die Hardware des Testaufbaus

Die in diesem Kapitel angeführte Hardware wurde zur Realisierung des Testaufbaus verwendet. Dieses Material wurde von der Firma Wago aus Wien (Feldbus-Controller inkl. diverser Busklemmen und serielles Anschlusskabel) und von der Firma IGEA aus Herzogenburg (24-Volt-Netzgerät, diverse elektrische Leitungen zum Testen und eine Tragschiene mit Hutprofil zur Befestigung des Feldbus-Controllers, der Busklemmen und des Netzgerätes) zur Verfügung gestellt. Die Störmeldeeingänge (digitale Eingänge) wurden mithilfe von Drahtbrücken simuliert.

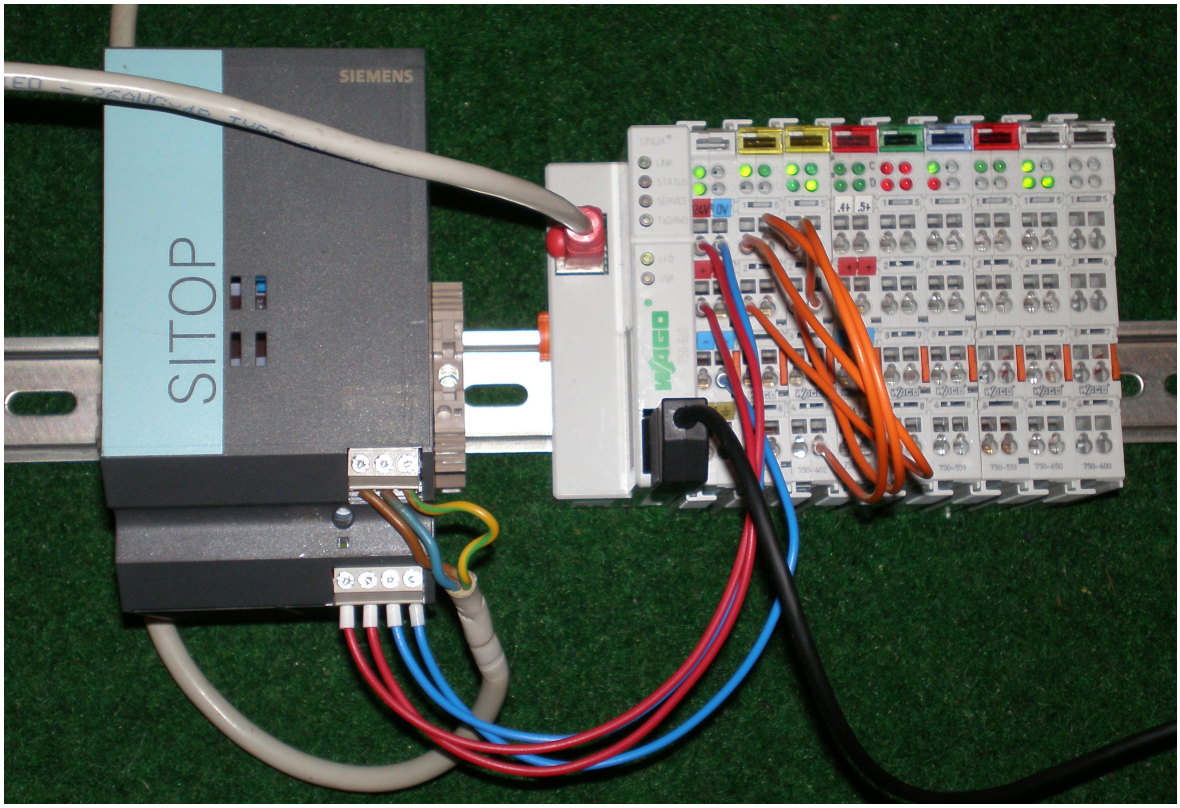

Abbildung 9: Der Testaufbau montiert auf einer Hutschiene bestehend aus Netzgerät und Feldbus-Controller mit Busklemmen

Dieses Kapitel beschreibt weiterführende Informationen und technische Daten zum Feldbus-Controller und dessen Busklemmen, welche für den Aufbau des Projekts zu beachten bzw. relevant sind. Dies ist ein wichtiger Teil in Dokumentationen zu elektrischen Anlagen.

Linux® Programmierbarer Feldbus-Controller 750-860

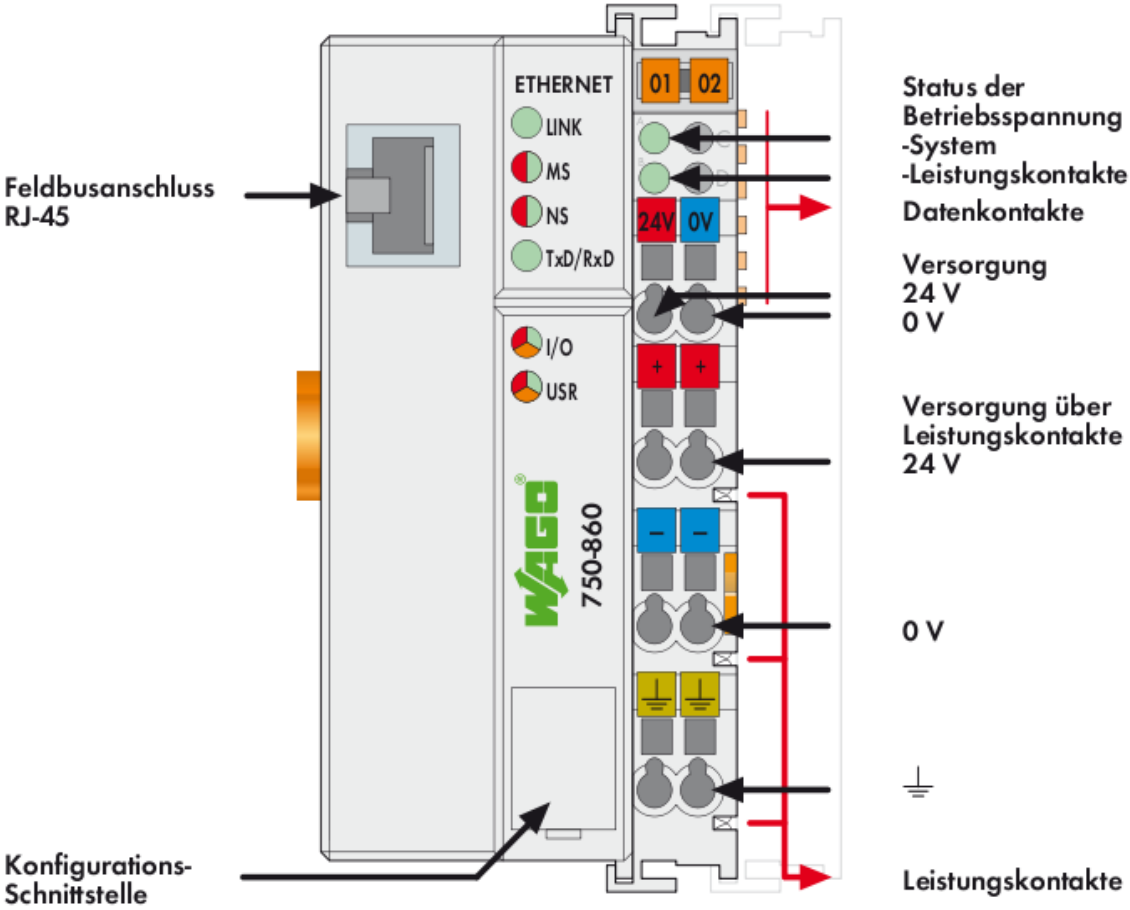

Abbildung 10: Anschlüsse des Feldbus-Controllers (WAGO 2008, S. 1)

Mit dem programmierbaren Linux Ethernet Feldbus-Controller werden Hochsprachen für die Software verwendet, anstatt die in der Automatisierungstechnik gebräuchlichen Programmiersprachen Anweisungsliste (AWL), Funktionsplan (FUP) und Kontaktplan (KOP). (vgl. Autor unbekannt 16. 4. 2009, 1, Die Basissprachen KOP, FUP, AWL, http://support.automation.siemens.com/WW/view/de/21062590)

Im Feldbus-Controller wird ein embedded Linux Betriebssystem mit der Kernel Version 2.6 zur effizienten Softwareentwicklung eingesetzt. Durch diese Gegebenheit erspart man sich spezielle und teure Hardware bzw. kleine PC Systeme bei den verschiedensten Applikationen. Aufgrund der dadurch vielseitig möglichen Fehlerquellen bietet der Hersteller allerdings nur für die Hardware einen Support an. (vgl. WAGO 2006, S. 46 ff.)

Zusätzlich zum Feldbus-Controller mit der Artikelnummer 750-860 sind in jedem Fall ein Konfigurationskabel (Artikelnummer: 750-920) und eine Abschlussklemme (Artikelnummer: 750-600) notwendig.

Anzahl der Controller am Master	limitiert durch ETHERNET- Spezifikation
Übertragungsmedium	Twisted Pair S-UTP 100 Ω Cat 5
max. Bussegmentlänge	100 m zwischen Hub und 750-860; max. Netzwerklänge durch ETHERNET Spezifikation limitiert
Übertragungsrate	10/100 Mbit/s
Busanschluss	RJ-45
Protokolle	MODBUS/TCP, HTTP, BootP, DHCP, DNS, SNTP, FTP, NFS

Tabelle 1: Systemdaten des Feldbus-Controllers 750-860

(vgl. WAGO 2008, S. 1)

CPU	32-Bit-Risc ARM7TDMI
RAM-Speicher	16 MByte SDRAM, 32 kByte NOVRAM
FLASH	4 MByte
EEPROM	4 kByte
Betriebssystem	Linux (Kernelversion 2.6)
Spannungsversorgung	DC 24 V (-15 % ... +20 %)
Eingangsstrom max. (24 V)	500 mA
Anzahl Busklemmen	64
mit Busverlängerung	250
Eingangsprozessabbild Feldbus max.	2 kByte
Ausgangsprozessabbild Feldbus max.	2 kByte
Abmessungen (mm) B x H x T	51 x 65 x 100
Gewicht	195 g
Schutzart	IP 20

Tabelle 2: Technische Daten des Feldbus-Controllers 750-860

(vgl. WAGO 2008, S. 2)

Busklemme 750-402 mit 4 digitalen Eingängen

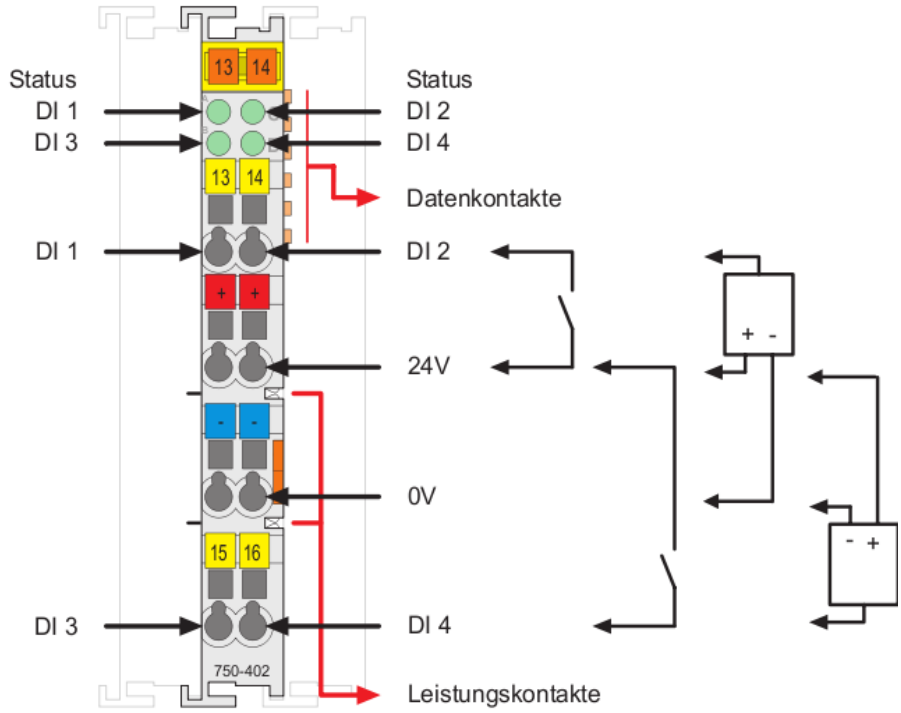

Abbildung 11: Busklemme 750-402 mit 4 digitalen Eingängen (WAGO 2009, S. 9)

Diese Busklemme mit 4 digitalen Eingängen wird für die Störmeldeeingänge des Summenstörmeldesystems verwendet und mit den potentialfreien Kontakten der zu überwachenden Geräte verschalten.

Artikelnummer: 750-402 (WAGO Kontakttechnik GmbH & Co. KG)
Bezeichnung: 4 DI DC 24 V 3,0 ms
Beschreibung: 4-Kanal Digital Eingangsklemme DC 24 V 3,0 ms, 2- oder 3-Leiter Anschluss; positiv schaltend

Zur Störunterdrückung ist jedem Eingang ein RC-Filter mit einer Zeitkonstanten von 3,0 ms vorgeschaltet. Der Signalzustand wird von einer Status-LED pro Eingang angezeigt.

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2009, S. 9 ff.)

Stromaufnahme (intern)	7,5 mA
Signalspannung (0)	DC -3 V bis +5 V
Signalspannung (1)	DC 15 V bis 30 V
Eingangsstrom typ.	4,5 mA
Potentialtrennung	500 V (System/Feld)
Datenbreite intern	4 Bit
Eingangsbit: B3 / B2 / B1 / B0	Signalzustand: DI4 / DI3 / DI2 / DI1

Tabelle 3: Technische Daten der Busklemme 750-402 mit 4 digitalen Eingängen (vgl. WAGO 2009, S. 12 f.)

Busklemme 750-504 mit 4 digitalen Ausgängen

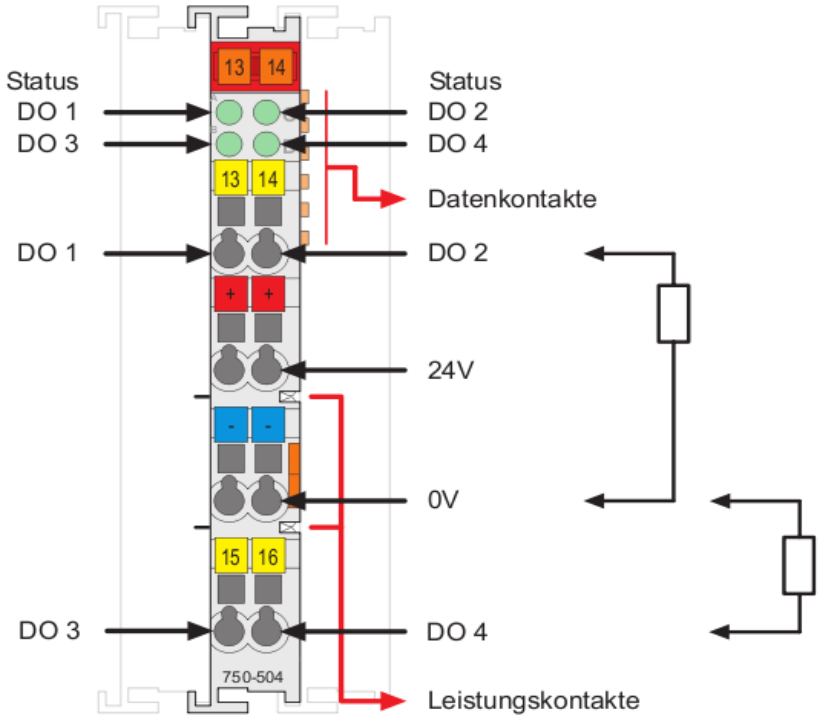

Abbildung 12: Busklemme 750-504 mit 4 digitalen Ausgängen (WAGO 2009a, S. 11)

Ein digitaler Ausgang wird zur Ansteuerung der lokalen Summenstörmeldelampe verwendet.

Artikelnummer: 750-504 (WAGO Kontakttechnik GmbH & Co. KG)
Bezeichnung: 4 DO DC 24 V 0,5 A, positiv schaltend
Beschreibung: 4-Kanal-Digitalausgangsklemme DC 24 V 0,5 A kurzschlussfest, positiv schaltend

Für den Anschluss induktiver Lasten ist parallel zum Verbraucher eine angepasste Schutzbeschaltung, z. B. eine Freilaufdiode, einzusetzen. Der Signalzustand wird von einer Status-LED pro Ausgang angezeigt.

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2009a, S. 11 ff.)

Stromaufnahme (intern)	7 mA
Stromaufnahme typ. (Feldseite)	30 mA (pro Modul) + Last
Lastart	ohmsch, induktiv, Lampenlast
Schaltfrequenz max.	1 kHz
Verpolungsschutz	ja
Ausgangsstrom	0,5 A kurzschlussfest
Potentialtrennung	500 V (System/Feld)
Datenbreite intern	4 Bit
Ausgangsbit: B3 / B2 / B1 / B0	Signalzustand: DO4 / DO3 / DO2 / DO1

Tabelle 4: Technische Daten der Busklemme 750-504 mit 4 digitalen Ausgängen (vgl. WAGO 2009a, S. 14 f.)

Busklemme 750-459 mit 4 analogen Eingängen

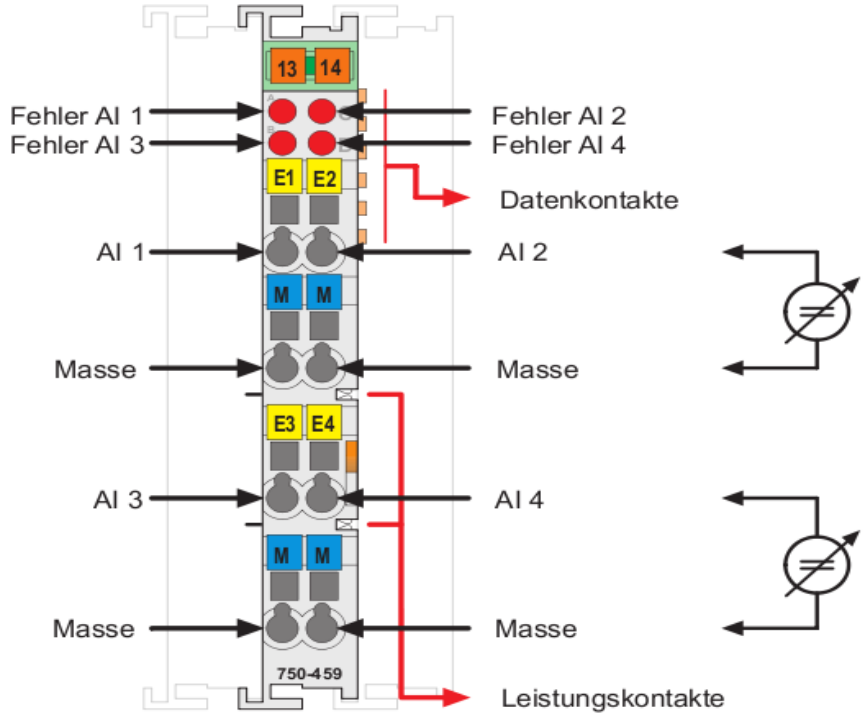

Abbildung 13: Busklemme 750-459 mit 4 analogen Eingängen (WAGO 2009b, S. 7)

Zum Vergleich mit den digitalen Busklemmen wird hier eine analoge Busklemme angeführt. Vor allem das Prozessabbild unterscheidet sich wesentlich von den digitalen Busklemmen.

Artikelnummer: 750-459 (WAGO Kontakttechnik GmbH & Co. KG)
Bezeichnung: 4 AI DC 0-10 V, Single-Ended
Beschreibung: 4-Kanal Analog Eingangsklemme (0-10 V, Single-Ended)

Alle Masseanschlüsse sind zusammengefasst und liegen auf dem 0V-Massepotenzial der letzten davor befindlichen Einspeiseklemme der Feldversorgung auf. Die Eingangssignale werden mit 12 Bit aufgelöst. Die Fehler-LED signalisiert, dass sich der jeweilige Messwert außerhalb des zulässigen Messwertes befindet.

Pro Messkanal werden 16 Bits für den Messwert und 8 Bits für den Status ausgegeben. Dieses Datenwort (16 Bit) wird im Prozessabbild des Feldbuscontrollers abgelegt. Der Messwert von 12 Bit wird dabei auf Bit 3 bis 14 abgebildet. Das Vorzeichen-Bit (Bit 15) wird bei dieser Busklemme nicht verwendet.

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2009b, S. 7 ff.)

Spannungsversorgung	über Systemspannung DC/DC
Stromaufnahme typ. (intern)	65 mA
Eingangsspannung max.	± 40 V
Signalspannung	0V ... 10 V
Eingangswiderstand typ.	> 100 kΩ
Auflösung	12 Bit
Wandlungszeit typ.	10 ms
Messfehler 25°C	<± 0,1 % vom Skalenendwert
Datenbreite	4 x 16 Bit Daten 4 x 8 Bit Steuer / Status (optional)

Tabelle 5: Technische Daten der Busklemme 750-459 mit 4 analogen Eingängen (vgl. WAGO 2009b, S. 9)

Eingangs-spannung in Volt	Zahlenwert				Fehler LED
	Binär		Hex.	Dezimal	
	Messwert	X F Ü *			
< 0	'0000.0000.0000.0	X11'	0x0003	3	Ein
0	'0000.0000.0000.0	X00'	0x0000	0	Aus
1	'0000.1100.1100.1	X00'	0x0CCC	3276	Aus
2	'0001.1001.1001.1	X00'	0x1998	6552	Aus
5	'0100.0000.0000.0	X00'	0x4000	16384	Aus
9	'0111.0011.0011.0	X00'	0x7330	29488	Aus
10	'0111.1111.1111.1	X00'	0x7FFC	32764	Aus
> 10	'0111.1111.1111.1	X11'	0x7FFF	32767	Ein

Tabelle 6: Prozessdaten der Busklemme 750-459 mit 4 analogen Eingängen (Statusbits: X = nicht benutzt, F = Kurzschluss, Ü = Überlauf) (vgl. WAGO 2009b, S. 12)*

Weitere Hardwarekomponenten

Zu den in diesem Kapitel angeführten Materialien waren am Testaufbau folgende Geräte zusätzlich vorhanden:

- Netzgerät: Siemens SITOP (6EP1 334-2BA01) 24V 10A
- Wago Busklemme 750-559: 4-Kanal Analog Ausgangsklemme 0-10 V
- Wago Busklemme 750-401: 2-Kanal Digital Eingangsklemme; DC 24 V; 0,2 ms;
- Wago Busklemme 750-513: 2-Kanal Relaisausgangsklemme; Relais 2 NO / Potentialfrei; AC 230 V, DC 30 V;
- Wago Busklemme 750-650: Serielle Schnittstelle; RS 232 C / 9600 / N / 8 / 1;
- Wago Busklemme 750-600: Bus-Endklemme; für TS 35;

3.3.2 Client-Rechner

Der Webbrowser sendet einen HTTP-Request an den Webserver. Im Response des Webservers ist auch Javascript enthalten, welches sich automatisch im Hintergrund um das Nachladen von geänderten Datensätzen bzw. Störmeldungen kümmert. Weiters kann man vom Browser aus aktive Störmeldungen in der Datenbank am Webserver quittieren. Dabei werden Datensätze verändert, der Zeitstempel aktualisiert, aktive Polling-Skripte terminieren mit einer Meldung an deren Webbrowser und diese laden aufgrund der Meldung die geänderte Störmeldeliste nach.

Abbildung 14: Ansicht des Webbrowsers mit der Summenstörmeldeseite

Folgende Elemente bzw. Informationen werden auf der Webseite der Summenstörmeldungen angezeigt:

- „Die letzte Aktualisierung erfolgte:“ bezieht sich auf den Zeitpunkt des letzten Ladevorgangs vom Webserver. Dieser Zeitpunkt wird verändert bzw. erstellt wenn die Webseite aufgerufen wird, die Liste per „Liste aktualisieren“-Button manuell abgerufen wird oder die Liste aufgrund einer Änderung in der Datenbank neu geladen wird.

- „Die letzte Änderung erfolgte:“ bezieht sich auf den Zeitpunkt der letzten Änderung in der Datenbank aufgrund einer Änderung bei den, vom Feldbus-Controller übertragenen, Störmeldungen.

- Der „Liste aktualisieren“-Button lädt die Daten vom Webserver erneut herunter. Diese Funktion wird auch von Javascript genutzt, allerdings nur wenn eine Änderung der Daten in der Datenbank statt gefunden hat. Der Button selbst ist nur für Testzwecke vorhanden und für einen produktiven Einsatz nicht erforderlich.

- Der „Störungen quittieren“-Button schickt diese Anforderung an den Webserver. In der Datenbank wird der Status von aktiven (rot markierten) Störmeldungen in Quittiert (gelb markiert) geändert. Danach wird vom Javascript die aktualisierte Störmeldeliste vom Webserver angefordert.

- Die Störmeldeliste enthält aktive (rote), quittierte (gelbe) und „OK“ (grüne) Meldungen und wird mittels Javascript mit den Daten des Webservers bis auf wenige Sekunden Verzögerung synchron gehalten.

Die Sortierung der Störmeldeliste erfolgt in erster Linie nach dem Störmeldestatus, damit in langen Listen nicht nach aktiven Störungen gesucht werden muss, sondern diese immer am oberen Ende der Liste angezeigt werden.

Auszug aus dem Quellcode der Datei index.html

Um den Javascriptcode bzgl. Ajax crossbrowserfähig zu halten, wird das Javascript Framework Prototype eingesetzt.

```
<script src="js/prototype-1.6.0.3.js" type="text/javascript"></script>
```

Listing 15: index.html: Einbinden des Prototype Frameworks

```
// Prototype: verschiedene Eventhandler registrieren
Ajax.Responders.register({
  onException: function(){
    alert('An exception occured!');
  },
  onFailure: function(){
    alert('A failure happens!');
  },
  onComplete: function(){
    setLoadTextFalse();
  }
});
```

Listing 16: index.html: Eventhandler des Prototype Frameworks setzen

Manuelles Laden der Daten vom Webserver wird vom „Liste aktualisieren“ Button ausgelöst

```
            // Daten für den Anwender sichtbar aktualisieren
            function manRefresh () {
                  loadText = true;
                  loadTextF("1");
                  setTimeout("loadMessages();",2000);
            }
            // Prototype: Daten per Ajax holen und im HTML-Element per
ID ersetzen
            function loadMessages () {
                  new Ajax.Updater('messages', 'loadMessages.php',
{ method: 'get' });
            }
```

Listing 17: index.html: Manuelles Laden der Daten vom Webserver

Das Quittieren aktiver Störmeldungen wird am Webserver durch Aufruf der entsprechenden PHP Datei ausgeführt.

```
// Quittierung, für den Anwender sichtbar
function manQuitt () {
      loadText = true;
      loadTextF("1");
      setTimeout("quittMessages();",1000);
}
// Prototype: Quittierungsbefehl an Webserver übergeben
function quittMessages () {
      new Ajax.Request('quittMessages.php', {
            method: 'get',
            onSuccess: quittResponse
      });
}
// Prototype: Nach erfolgter Quittierung Daten neu laden
function quittResponse (responseObject) {
            setTimeout("manRefresh();",100);
}
```

Listing 18: index.html: Quittieren aktiver Störmeldungen

Die Funktion „updateMessages“ übergibt der aufgerufenen PHP Datei den aktuellen Zeitstempel der lokalen Daten. Ändern sich die Daten am Webserver, wird sofort eine entsprechende Antwort gesendet. Nach etwa 50 Sekunden wird in jedem Fall eine Antwort zurück geschickt, um kein Timeout des Webbrowsers zu verursachen. Im Firefox lässt sich dieses Verhalten über den Parameter „network.http.request.timeout“ in der Config ändern. Allerdings müsste man dann diese Einstellung auf allen Webbrowsern durchführen, welche die Summenstörmelde-Webseite anzeigen sollen. Andernfalls wird der Webbrowser mit einer Fehlermeldung nach Ablauf des Timeouts abbrechen.

```
// Erfolgt keine Änderung der Störmeldungen, kommt eine Antwort erst
nach über 50 Sekunden
function updateMessages () {
      try {
               var reqPara =
'timeStamp='+document.getElementById('timeStamp').innerHTML;
      } catch (e) {
               setTimeout("updateMessages();",3000);
               return;
      }
      new Ajax.Request('updateMessages.php', {
               method: 'post',
               parameters: reqPara,
               onSuccess: updateResponse
      });
}
// Antwort vom Server: bei Veränderung Daten laden, ansonsten wieder
Anfrage auf Veränderung
function updateResponse (responseObject) {
      if ( />OK!</.test(responseObject.responseText) ) {
//alert(responseObject.responseText+' -> Keine Veränderung, weiter
beobachten ...');
               setTimeout("updateMessages();",500);
      } else {
//alert(responseObject.responseText+' -> Veränderung, NEU LADEN!!!');
               setTimeout('setFirstRun(true)',100);
               setTimeout("manRefresh();",500);
      }
}
```

Listing 19: index.html: Anfrage an Webserver nach Änderungen

Die Anzeige des „Loading ...“ Textes erfolgt animiert, um beim Anwender kein Gefühl des Abbruchs bzw. „hängen bleiben“ der Anwendung zu vermitteln. Dies kann passieren, weil die Daten im Hintergrund geladen werden und dadurch für den Anwender keine Anzeichen für diesen Vorgang zu erkennen wären.

```
function loadTextF (nr) {
      if (loadText == true) {
            chAr = new Array["|", "/", "-", "\"];
            document.getElementById("messages").innerHTML =
"<br/><br/><br/><h3>"+ chAr[nr] +" Loading ... "+ chAr[nr] +"</h3>";
            var neu = (nr + 1) % 4;
            setTimeout("loadTextF("+neu+");",100);
      }
}
```

Listing 20: index.html: Anzeige des animierten „Loading ...“ Textes

Der Großteil der Webseite wird am Webserver mit PHP erstellt und in in das „div“ Element eingefügt. Mithilfe der ID des Elements und des Prototype Frameworks ist das mit geringem Aufwand möglich.

```
<BODY LANG="de-AT" onload="manRefresh();" style="text-align:center;">
      <h1>Summenstörmeldungen</h1>
      <div id="messages" align="right" style="text-align:center;
width:auto; margin: 10px;"></div>
</BODY>
```

Listing 21: index.html: „Body“ mit „div“ in das die Störmeldeliste eingefügt wird

(vgl. Autor unbekannt 3. 4. 2009, 1, Introduction to Ajax, http://www.prototypejs.org/learn/introduction-to-ajax)

3.3.3 Webserver

Der Webserver ist das zentrale Element dieses Projektaufbaus. Der Großteil der Logik ist auf dem Webserver implementiert. Sämtliche Anfragen der Webbrowser für die Summenstörmelde-Webseite werden hier beantwortet. Auch die Kommunikation mit dem Feldbus-Controller wird vom Webserver erledigt.

MySQL Datenbank

Sämtliche Daten vom Feldbus-Controller inkl. Zeitstempel werden in dieser Datenbank abgelegt. In einer Tabelle sind sämtliche Störmeldungen mit Störmeldenummer, Störmeldestatus und Störmeldebeschreibung abgelegt. Beim Einsatz mehrerer Feldbus-Controller sollte die Tabelle um die Spalte „controllerNumber“ erweitert werden.

```
chris@chrux:~$ mysql --user=ssm –password=kjh... ssm -e "SELECT * FROM
messages" -h chriswind.home...
+---------------+---------+-----------------------------------------------+
| messageNumber | zustand | messageText                                   |
+---------------+---------+-----------------------------------------------+
|             1 |       2 | Störmeldung 1: Erste Klemme, Erster Eingang   |
|             2 |       0 | Störmeldung 2: Erste Klemme, Zweiter Eingang  |
|             3 |       2 | Störmeldung 3: Zweite Klemme, Erster Eingang  |
|             4 |       1 | Störmeldung 4: Zweite Klemme, Zweiter Eingang |
|             5 |       1 | Störmeldung 5: Zweite Klemme, Dritter Eingang |
|             6 |       2 | Störmeldung 6: Zweite Klemme, Vierter Eingang |
+---------------+---------+-----------------------------------------------+
```

Listing 22: "ssm"-DB: "messages"-Table

In einer zweiten Tabelle wird der Zeitstempel der letzten Änderung der Daten hinterlegt. Dieser Zeitstempel wird mit dem in der Anfrage des Clients mitgeschickten Zeitstempel verglichen. Falls diese nicht gleich sein sollten werden die Daten vom Client neu angefordert.

```
chris@chrux:~$ mysql --user=ssm –password=kjh... ssm -e "SELECT * FROM
lastChange" -h chriswind.home...
+---------+---------------------+-------+
| indexNr | timestamp           | value |
+---------+---------------------+-------+
|       1 | 2009-04-02 17:50:23 |     1 |
+---------+---------------------+-------+
```

Listing 23: "ssm"-DB: "lastChange"-Table

Die Änderung des Zeitstempels wird mithilfe eines sogenannten Triggers durchgeführt. Ein Trigger wird mittels eines SQL Statements erzeugt und benötigt grundsätzlich fünf Informationen:

1. Der Trigger-Name zur Verwaltung in der Datenbank als Objekt.
2. Die Trigger-Time kann BEFORE oder AFTER sein, je nachdem ob das Trigger-Statement vor oder nach der Anweisung durchgeführt werden soll, durch die der Trigger ausgelöst wird.
3. Der Tabellenname bezieht sich auf jene Tabelle, durch die der Trigger ausgelöst wird.
4. Das Trigger-Event bezeichnet die Art des Statements, durch die der Trigger ausgelöst wird (INSERT, UPDATE oder DELETE).
5. Das Trigger-Statement enthält die Anweisung(en), welche durch den Trigger ausgeführt werden sollen.

```
DROP TRIGGER renewTimeStamp;
CREATE TRIGGER renewTimeStamp
      AFTER UPDATE ON messages
      FOR EACH ROW
      UPDATE lastChange SET value = (value + 1) % 2 WHERE indexNr = 1;
```

Listing 24: "ssm"-DB: "renewTimeStamp"-Trigger

(vgl. Autor unbekannt 3. 4. 2009, 1, CREATE TRIGGER, http://dev.mysql.com/doc/refman/5.1/de/create-trigger.html)

Apache Webserver mit PHP Erweiterung

Der Webserver nimmt die Anfragen von den Webbrowsern der Clients entgegen. Die Antwort wird mithilfe von PHP generiert, welches die Daten von der MySQL Datenbank abfragt, diese Daten aufbereitet, das Ergebnis als HTML-Fragment in die Antwort einpackt und an den Client retour sendet.

Im Fall einer Anfrage für die Datei „loadMessages.php“ werden alle Störmeldungen inkl. dem Zeitstempel von der Datenbank abgefragt und für das Einfügen in das „div“-Element mit der ID „messages“ im Webbrowser vorbereitet.

```
...
$db = @new mysqli('localhost', 'ssm', '***', 'ssm');
if (mysqli_connect_errno()) {
    die ('Konnte keine Verbindung zur Datenbank aufbauen:
'.mysqli_connect_error().'('.mysqli_connect_errno().')');
}

$sql = 'SELECT `timestamp` FROM `lastChange` LIMIT 0,1;';
$result = $db->query($sql);
if (!$result) {
    die ('Etwas stimmte mit dem Query nicht: '.$db->error);
}

$row = $result->fetch_assoc();
...
echo '<tr><td style="text-align:right;">Die letzte Aktualisierung
erfolgte: </td><td>'.date("Y-m-d H:i:s").'</td></tr>';
echo '<tr><td style="text-align:right;">Die letzte Änderung erfolgte:
</td><td id="timeStamp">'.$row['timestamp'].'</td></tr>';
...
$sql = 'SELECT * FROM `messages` ORDER BY `zustand`, `messageNumber`
ASC;';
$result = $db->query($sql);
if (!$result) {
    die ('Etwas stimmte mit dem Query nicht: '.$db->error);
}
...
while ($row = $result->fetch_assoc()) {  // NULL ist äquivalent zu
false
      if ($row['zustand'] == 2) {
         $color = "9BFF9B"; // OK:(dezentes)grün
            $stateText = "OK";
      } elseif ($row['zustand'] == 1) {
         $color = "FFFF00"; // Quitt:gelb
            $stateText = "Quittiert";
      } else {
         $color = "FF0000"; // Stoer:rot
            $stateText = "STÖRUNG";
      }
      echo '<tr id="tr'.$row['messageNumber'].'" style="background-
color:#'.$color.';">';
      echo '<td id="tdNumber">'.$row['messageNumber'].'</td>';
      echo '<td id="tdState">'.$stateText.'</td>';
      echo '<td
id="tdText">'.htmlentities($row['messageText']).'</td>';
      echo '</tr>';
}
...
```

Listing 25: Webserver: loadMessages.php

Bei einer Anfrage nach der Datei „updateMessages.php“ wird vom PHP Skript der Zeitstempel von der Datenbank und der vom Client auf Gleichheit überprüft. Diese Überprüfung wird solange durchgeführt, bis entweder die beiden Zeitstempel nicht mehr übereinstimmen oder die Überprüfung 27 Mal durchgeführt wurde. Dies entspricht mindestens 54 Sekunden und soll ein Timeout des Webbrowsers verhindern.

```
<?php
$db = @new mysqli('localhost', 'ssm', '***', 'ssm');
if (mysqli_connect_errno()) {
    die ('Konnte keine Verbindung zur Datenbank aufbauen:
'.mysqli_connect_error().'('.mysqli_connect_errno().')');
}

for ($i = 0; $i < 27; $i++) {
      sleep(2);
      $sql = 'SELECT `timestamp` FROM `lastChange` LIMIT 0,1;';
      $result = $db->query($sql);
      if (!$result) {
          die ('Etwas stimmte mit dem Query nicht: '.$db->error);
      }
      $row = $result->fetch_assoc();
      echo $i.',';
      if ($row['timestamp'] !== $_POST['timeStamp']) {
              return;
      }
}
echo '>OK!<';
?>
```

Listing 26: Webserver: updateMessages.php

Beim Betätigen des Quittierungs-Button durch den Anwender im Webbrowser wird die Datei „quittMessages.php“ am Webserver aufgerufen und ändert den Status in der Datenbank von aktiven (rot) Störmeldungen auf quittiert (gelb). Danach ruft der Webbrowser die Datei „loadMessages.php“ am Webserver auf, um seine Daten für die Anzeige aktualisieren zu können.

```
<?php
$db = @new mysqli('localhost', 'ssm', '***', 'ssm');
if (mysqli_connect_errno()) {
    die ('Konnte keine Verbindung zur Datenbank aufbauen:
'.mysqli_connect_error().'('.mysqli_connect_errno().')');
}

$sql = "UPDATE messages SET zustand = 1 WHERE zustand = 0 ; ";
$result = $db->query($sql);
if (!$result) {
    die ('Etwas stimmte mit dem Query nicht: '.$db->error);
}

?>
```

Listing 27: Webserver: quittMessages.php

Der Feldbus-Controller schickt aktuelle Informationen seiner Busklemmen, an denen die Störmeldeausgänge der zu überwachenden Geräte angeschlossen sind, als JSON-String an den Webserver. In diesem Fall ist die Datei „updateDI.php“ zuständig und diese speichert die Informationen unter Berücksichtigung des bisherigen Störmeldezustands in der Datenbank ab, da der Feldbus-Controller keine Kenntnis über bereits quittierte Störmeldungen hat.

```
<?php
// Json String vom Feldbus-Controller in ein PHP Objekt umwandeln
$stCl = json_decode( stripslashes($_POST["json"]) );
$arrDI = $stCl->{'di'};

$db = @new mysqli('localhost', 'ssm', 'rWXQY8P2WDw6chaz', 'ssm');
if (mysqli_connect_errno()) {
    die ('Konnte keine Verbindung zur Datenbank aufbauen:
'.mysqli_connect_error().'('.mysqli_connect_errno().')');
}

$sql = 'SELECT `messageNumber`, `zustand` FROM `messages` ORDER BY
`messageNumber` ASC;';
$result = $db->query($sql);
if (!$result) {
    die ('Etwas stimmte mit dem Query nicht: '.$db->error);
}

while ($row = $result->fetch_assoc()) {  // NULL ist äquivalent zu
false
      $meNrStr = $row['messageNumber'];
      if ($arrDI[$row['messageNumber']-1]) { // Wenn DI am FBK true ...
              $zustStr = "2"; // ... Zustand auf OK setzen
      }
      else { // Wenn DI am FBK false ...
              if ($row['zustand'] == 1) {
                 $zustStr = "1"; // ... Zustand bleibt auf Quitt
              }
              else { // zustand == 0 oder 2
                 $zustStr = "0"; // ... Zustand auf Stoerung setzen
              }
      }
      $sql2 = "UPDATE messages SET zustand = '".$zustStr."' WHERE
messageNumber = ".$meNrStr." ; ";
      echo $sql2."\n";
      $result2 = $db->query( $sql2 );
      if (!$result2) {
          die ('Etwas stimmte mit dem Query nicht: '.$db->error);
      }
}
?>
```

Listing 28: Webserver: updateDI.php

(vgl. Autor unbekannt 3. 4. 2009, 1, The MySQL Native Driver,

http://at2.php.net/manual/de/mysqli.mysqlnd.php)

(vgl. Autor unbekannt 3. 4. 2009, 1, Einführung in JSON, http://www.json.org/json-de.html)

3.3.4 Feldbus-Controller

Der Feldbus-Controller überwacht seine Störmeldeeingänge auf Veränderungen. Tritt eine Änderung ein, werden die Daten per HTTP-Request an den Webserver gesendet und dieser übernimmt das Eintragen der Daten in die Datenbank.

Dieses in C geschriebene Programm liest mithilfe des Kbus-Treibers die digitalen Eingangsklemmen aus und setzt eine Ausgangsklemme als Summenstörmeldung, um auch Vorort eine optische und/oder akustische Signalisierung realisieren zu können. Wenn sich das Prozessabbild der digitalen Eingangsklemmen geändert hat, wird eine HTTP-Nachricht erstellt und per TCP an den Webserver an Port 80 geschickt.

Mit dem Kbus-Treiber werden über das Prozessabbild und den Klemmenbus alle Busklemmen angesprochen. Diese können dadurch ausgelesen bzw. gesetzt werden, je nachdem ob es sich um einen Eingang oder Ausgang handelt. Busklemmen gibt es wie bereits angeführt in verschiedenen Ausführungen: digitale Ein- und Ausgangsklemmen, analoge Ein- und Ausgangsklemmen, sowie diverse Sonderklemmen.

Zu Beginn des Programms werden Includes und Variablen definiert und zum Teil auch Werte zugewiesen. Der HTTP-Header ist bis auf den Wert der „Content-length“ bereits vorgegeben. Dieser Wert wird vor der Übertragung an den Webserver eingefügt, da sich dieser erst durch den Zusammenbau des JSON-Strings ergibt.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
#include <sys/types.h>
#include <asm/types.h>
#include <string.h>
#include "kbusapi.h"

#ifndef __LINUX_KBUS_H__
#define __LINUX_KBUS_H__ 1
#endif

// Es wird ein Byte an Digitalen Eingaengen verarbeitet
#define ANZAHL_DI 8

int main (int argc, char **argv)
{
  FILE *fpipe; // Pointer auf I/O Stream
  int BinOffset, PabDigIn, lastChangeB1, i, n=0;
  // Fuer Poststring inkl. Anfangssequenz
  char json[(ANZAHL_DI*8)+15];
  // HTTP-Header zur Datenuebergabe an den Webserver
  static char httpRequest[(220+(ANZAHL_DI*8))];
  static const char header[]={
    "echo \""
    "POST /updateDI HTTP/1.1\n"
    "Host: ssm.chriswind.homelinux.com\n"
    "Connection: close\n"
    "User-Agent: CW+NC\n"
    "Content-type: application/x-www-form-urlencoded\n"
    "Content-length:"
  };

  lastChangeB1 = 0; // Wert der letzten Aenderung
  sleep(3);
```

Listing 29: C-Programm am Feldbus-Controller, Teil 1 von 4

Am Anfang der Endlosschleife wird der Kbus-Treiber (siehe Listings 10 und 11) mithilfe der Kbus-API (siehe Listing 9) geöffnet, das Prozessabbild aktualisiert und die Eingänge abgefragt. In Abhängigkeit der Eingänge wird der Summenstörmeldeausgang mithilfe von Bit-Operationen entweder gesetzt oder zurückgesetzt. Danach wird wieder das Prozessabbild aktualisiert, damit auch der digitale Ausgang vom Treiber verändert wird und der Treiber selbst geschlossen wird.

```
  while (1)
  {
    sleep(1); // Ist ja keine Echtzeit-Anwendung!
    // Eingangsklemmen mittels Klemmenbustreiber abfragen
    KbusOpen();
    KbusUpdate();
    // Offset für Digitale Eingaenge ermitteln
    BinOffset = KbusGetBinaryInputOffset() / 8;
    // Aktuelles Prozessabbild abfragen
    PabDigIn = (int)pstPabIN->uc.Pab[BinOffset];

    // Offset für Digitale Ausgaenge ermitteln
    BinOffset = KbusGetBinaryOutputOffset() / 8;
    // Summenstoermeldung am Digitalen Ausgang 12.0 ausgeben
    if ( (PabDigIn % 8) < 7) { // die ersten drei Eingangsbits
vorhanden?
      // Mit bitweiser „oder“-Verknuepfung den ersten DA als
            Summenstoermeldung setzen
      pstPabOUT->uc.Pab[BinOffset] = pstPabOUT->uc.Pab[BinOffset] | 1;
    } else {
      // Mit bitweiser Und-Verknuepfung inkl. Einerkomplement den
ersten DA zuruecksetzen
      pstPabOUT->uc.Pab[BinOffset] = pstPabOUT->uc.Pab[BinOffset] & ~1;
    }
    KbusUpdate(); // Aenderung zurueck ins Prozessabbild schicken
    KbusClose();
```

Listing 30: C-Programm am Feldbus-Controller, Teil 2 von 4

Falls sich das Prozessabbild seit der vorhergehenden Abfrage verändert hat oder sich innerhalb von 50 Abfragen des Prozessabbildes nichts geändert hat, wird der JSON-String anhand des aktuellen Prozessabbildes erstellt, dessen Länge ermittelt und dieser Wert nach dem HTTP-Header eingefügt. Danach wird der JSON String angehängt. In Kombination mit dem Echo-Befehl, der Pipe und dem netcat-Befehl inkl. Serveradresse und Portnummer entsteht daraus eine Shell-Anweisung.

```
    n++;
// zum Testen
printf("if (akt:%d != lCh:%d) || ( %d >50)\n",PabDigIn,lastChangeB1,n);
    // Veraenderung im Prozessabbild oder 50x ohne Veraenderung?
    if ( (PabDigIn != lastChangeB1) || (n > 50) )
    {
      // n = 0; => Am Ende der Schleife, nur wenn Request "200 OK"
           war!!!
      // den String fuer strcat() als leer markieren
      *json = '\0';
      // Boolean mittels Bitoperation abfragen und dem Poststring
           anhaengen
      for (i = 1; i <= 128; i*=2)
        (PabDigIn & i) ? strcat(json,"true%2C") : strcat(json,"false
           %2C");
      // letztes Komma (URL-codiert) entfernen, Poststring abschliessen
      json[strlen(json)-3] = '\0';
      // Header, Poststringlaenge und Poststring zusammenbauen
      sprintf(httpRequest,"%s %d\n\njson={%%22di%%22%3A[%s]}\n\" | nc
-w 12 192.168.1.222 80\n",header,(strlen(json)+20),json);
      lastChangeB1 = PabDigIn;
// zum Testen
printf("%s",httpRequest);
```

Listing 31: C-Programm am Feldbus-Controller, Teil 3 von 4

Diese Shell-Anweisung wird dem System zur Ausführung übergeben und dabei eine lesende Pipe zum neuen Prozess geöffnet. Über diese Pipe wird die Antwort vom Webserver ausgelesen. Beinhaltet diese Antwort einen „200“-String und einen „OK“-String, dann war die Übertragung erfolgreich.

```
      if ( !(fpipe = (FILE*)popen(httpRequest,"r")) )
      {  // If fpipe is NULL
        perror("Problems with pipe");
        exit(1);
      }
      n=49;
      while (fgets(json, sizeof json, fpipe))
      {
// zum Testen
printf("<nc>%s", json);
        if ( strstr(json,"200") && strstr(json,"OK") )
          n=0; // Uebertragung erfolgreich, warten auf naechste
                  Aenderung
      }
      pclose(fpipe);
    }
  }
  exit(1);
}
```

Listing 32: C-Programm am Feldbus-Controller, Teil 4 von 4

Dieser C-Quellcode für den Feldbus-Controller wird mithilfe eines passenden Makefiles und dem GNU-Cross-Compiler der uC-Linux Distribution (arm-elf-Toolchain) kompiliert, anschließend per FTP oder NFS auf den Feldbus-Controller übertragen und kann dort ausgeführt werden.

(vgl. WAGO Kontakttechnik GmbH & Co. KG 2006, S. 100 ff.)

(vgl. Nikkanen 2003, S. 64 f.)

(vgl. Henning 2004, S. 52 f.)

(vgl. Autor unbekannt 3. 4. 2009, 1, Einführung in JSON, http://www.json.org/json-de.html)

3.4 Probleme bei der Projektumsetzung

Die arm-elf-Toolchain ist zum Kompilieren von Programmen bzw. des Betriebssystems für den Feldbus-Controller erforderlich. Diese Toolchain ließ sich auf dem Ubuntu-Rechner nicht auf Anhieb installieren weil:

- in den Installationsskripten „#!/bin/sh“ definiert ist und Ubuntu solche Skripte standardmäßig mit der Dash (Debian Almquist Shell) ausführt und diese mit einer Fehlermeldung abbricht.
- die Bash die Ausführung wegen eines Posix Versions Konflikts abgebrochen hat. Durch die Angabe von _POSIX2_VERSION=199209 vor dem Namen des Installationsskripts ließ sich auch dieses Problem lösen.
- im Board Support Package von der Firma Wago ist ein RPM Archiv enthalten. Mithilfe von „Alien“ konnte dieses in das DEB Format umgewandelt und installiert werden.

Das Kompilieren des uClinux OS und dessen Programme funktionierte nicht auf Anhieb, weil die „zlib“ auf dem Ubuntu Hostsystem nicht installiert war und dieser Umstand nicht eindeutig erkennbar war.

Falls ein zusätzliches Programm benötigt wird, welches nicht standardmäßig im Auslieferungszustand mit kompiliert wird, dann muss dessen Name im entsprechenden Makefile angegeben werden. Erst nach einem erneuten Kompiliervorgang ist ersichtlich, ob das jeweilige Programm mit dem Feldbus-Controller genutzt werden kann oder der Kompiliervorgang aufgrund eines Fehlers abbricht. Beim Versuch MySQL zu kompilieren wurde der Vorgang abgebrochen, weil die Hardwarearchitektur des Feldbus-Controllers von diesem Programm nicht unterstützt wird.

Es konnte per Ethernet keine Verbindung zum Feldbus-Controller lt. Handbuch aufgebaut werden, weil die Defaultwerte nicht eingestellt waren. Ein Verbindungsaufbau war daher nur über die serielle Schnittstelle mit einem speziellen Kabel für den Anschluss am Controller möglich.

Es war mir nicht möglich ein selbst erstelltes CGI Skript am Feldbus-Controller erfolgreich zum Laufen zu bringen. Der Boa-Webserver gab immer nur folgende Fehlermeldung über den Browser aus: „500 Server Error. The server encountered an internal error and could not complete your request.“ Für Servicezwecke wäre ein selbst erstelltes CGI-Skript, z.B. für die Ausgabe und das Löschen selbst erstellter Logs, eine hilfreiche Anwendung.

Dokumentation für die uClinux Distribution, deren Programme und Bibliotheken ist allgemein eher wenig vorhanden und so manche Information ist nur sehr schwer zu beschaffen, wie zum Beispiel für den Boa-Webserver. Vor allem wenn Controller-spezifische Probleme auftreten sind Lösungsansätze auch im Internet, wenn überhaupt, nur schwer zu finden.

3.5 Mögliche Erweiterungen

Folgende Erweiterungen bzw. Änderungen wären für dieses Projekt noch sinnvoll und wünschenswert:

- eine Einbindung von analogen Eingängen inklusive einer Eingabemöglichkeit von Schwellwerten zur Störmeldeauswertung. Damit könnte zum Beispiel der Füllstand eines Tanks individuell überwacht werden.
- Zugriff auf die Ein- und Ausgänge des Feldbus-Controllers mittels des Ethernet-Feldbuses „Modbus TCP“ über das Netzwerk.
- eine Loggingfunktion die jede Veränderung der Störmeldungen in der Datenbank mitprotokolliert. Ein möglicherweise hohes Datenvolumen sollte jedoch vermieden werden.
- eine Verwaltungsmöglichkeit (Editor) der Störmeldungen inkl. Beschreibung über einen Webbrowser. Damit könnten auch neue Störmeldungen schnell erstellt werden.
- Überwachung der Netzwerkverbindungen Webserver / Client und Webserver / Feldbus-Controller. Vor allem eine unterbrochene Verbindung zwischen Webserver / Feldbus-Controller könnte lange unbemerkt bleiben.
- Security Features zwischen Webbrowser und Webserver, wie zum Beispiel eine verschlüsselte Übertragung und Authentifizierung sind für manche Anwendungsfälle erforderlich.
- Evaluierung ob der Einsatz eines HTTP-Streams als Datenkanal zwischen Webbrowser und Webserver möglich bzw. sinnvoll ist. HTTP-Streams werden (noch) selten eingesetzt, da ein paar Vorraussetzungen erfüllt werden müssen. (vgl. Mahemoff 2006, S. 113 f.)
- die Interaktion mit dem Anwender durch visuelle Effekte wie Fading, Mutation oder Motion verbessern.

4 Fazit

Während die Installation der Entwicklungsumgebung auf einem Suse-Linux weitgehend problemlos ab lief, war die Installation auf einem Debian-System wesentlich aufwändiger. Aber diese Probleme konnten gelöst werden. Auch die Erstinbetriebnahme des Feldbus-Controllers verlief nicht reibungslos. Verstellte Defaultwerte waren hierfür der Grund.

Der Umgang mit dem Linux namens „uClinux“ ist natürlich nicht so komfortabel wie man dies von den Desktop-Versionen her gewöhnt ist. Für ein Microcontroller-Betriebssystem ist der Leistungsumfang jedoch enorm. Leider lassen sich nicht alle der verfügbaren uClinux-Programme für den Feldbus-Controller kompilieren. Eine Liste mit kompilierbaren Programmen wäre eine große Hilfe. Die Programme der uClinux-Distribution sind groß teils von den Desktop-Versionen her bekannt und deren Funktionsumfang inkl. Bedienung ist auch ähnlich.

Die C-Programmiersprache ist zwar nicht die komfortabelste Art zu programmieren, aber im Vergleich zur Anweisungsliste (AWL) mit deren Assembler-artigen Syntax ist die C-Programmierung wesentlich eleganter. Fertige technologische Funktionen, wie zum Beispiel ein Software-Regler bei Automatisierungsgeräten, stehen beim Feldbus-Controller allerdings nicht zur Verfügung. Auch die benötigte Zeit für eine Einarbeitung sollte nicht unterschätzt werden und die Eignung des Feldbus-Controllers für eine bestimmte Aufgabenstellung vorab sicher gestellt werden.

Die Stärke des Feldbus-Controllers liegt vor allem im Einsatz von kleineren Projekten, da man keine Kosten für Softwarelizenzen berücksichtigen muss. Die Kombination von Linux und Ethernet bietet einen großen Spielraum für die Kommunikation mit der Außenwelt. Ein Datenbank-Client würde aber meiner Ansicht nach die Einsatzmöglichkeiten des Feldbus-Controllers noch weiter verbessern.

5 Anhang

5.1 Anhang A: Literaturverzeichnis

5.1.1 Bücher

Burbiel, H. (2007): SOA & Webservices in der Praxis,
Poing: Franzis Verlag GmbH

Früh, K. F., Maier, U., Schaudel, D. (2008): Handbuch der Prozessautomatisierung: Prozessleittechnik für verfahrenstechnische Anlagen, 4. Auflage München: Oldenbourg Industrieverlag

Henning, P. A., Vogelsang, H. (2004): Taschenbuch Programmiersprachen,
München: Hanser Verlag GmbH

Langmann, R. (2007): Lean Web Automation in der Praxis,
Poing: Franzis Verlag GmbH

Langmann, R. (2004): Taschenbuch der Automatisierung,
München: Hanser Verlag GmbH

Mahemoff, M. (2006): Ajax Design Patterns,
Sebastopol: O'Reilly Media Inc.

Wellenreuther, G., Zastrow, D. (2008): Automatisieren mit SPS – Theorie und Praxis,
4. Auflage Wiesbaden: GWV Fachverlage GmbH

5.1.2 Zeitschriften

Lange, J. (2005): Zehn Jahre OPC: Von Data Access zu Unified Architecture,
In: SPS-Magazin 8/2005, Marburg: Technik-Dokumentations-Verlag GmbH

Tisken, V. (2008): Für individuelle Lösungen: Automatisierung mit Linux,
In: A&D Februar 2008, München: publish-industry Verlag GmbH

5.1.3 Handbücher und Datenblätter

WAGO Kontakttechnik GmbH & Co. KG (2006): Linux-Feldbuskoppler 750-860 , http://www.wago.com/wagoweb/documentation/750/ger_manu/860/m086000d.pdf

WAGO Kontakttechnik GmbH & Co. KG (2008): Linux® Programmierbarer Feldbus-Controller, http://www.wago.com/wagoweb/documentation/750/ger_dat/d086000d.pdf

WAGO Kontakttechnik GmbH & Co. KG (2009): Feldbusunabhängige Busklemmen: 4 DI DC 24 V 3,0 ms, positiv schaltend 750-402(/xxx-xxx), http://www.wago.com/wagoweb/documentation/750/ger_manu/modules/m040200d.pdf

WAGO Kontakttechnik GmbH & Co. KG (2009a): Feldbusunabhängige Busklemmen : 4 DO DC 24 V 0,5 A, positiv schaltend 750-504(/xxx-xxx), http://www.wago.com/wagoweb/documentation/750/ger_manu/modules/m050400d.pdf

WAGO Kontakttechnik GmbH & Co. KG (2009b): Feldbusunabhängige Busklemmen : 4 AI DC 0-10 V, Single-Ended 750-459, http://www.wago.com/wagoweb/documentation/750/ger_manu/modules/m045900d.pdf

WAGO Kontakttechnik GmbH & Co. KG (2009c): WAGO-I/O-IPC-G2 Linux 2.6: Industrie-PC mit Geode 266 Mhz , http://www.wago.com/wagoweb/documentation/758/ger_dat/d07580870001de.pdf

5.1.4 Bachelor's Thesis

Nikkanen, K. (2003): uClinux as an embedded solution, Turku Polytechnic Institute

5.1.5 Normen

ISA–The Instrumentation, Systems and Automation Society (1995): ANSI/ISA–88.01–1995 Batch Control Part 1: Models and Terminology, Research Triangle Park: ISA–The Instrumentation, Systems and Automation Society

5.1.6 Katalog

Siemens AG (2007): Katalog IK PI 2007, Add-ons 05/2007, Industrial Communication: PROFINET Development Kit DK-16xx PN IO

5.1.7 Präsentationen

Werner, Th., Pöschmann, A., Gnad, A., Krogel, P. (2005): Industrial Ethernet mit PROFINET v2.0 – Auch in embedded Linux Systemen, Barleben: ifak

5.1.8 Bilder

Siemens AG (2009a): Text Display TD 400C, http://www.automation.siemens.com/bilddb/download.asp?reqInsID=455395

Siemens AG (2009b): Mobile Panel 277F IWLAN, http://www.automation.siemens.com/bilddb/download.asp?reqInsID=456528

Siemens AG (2009c): THIN CLIENT PRO 15 Zoll mit Erweiterungseinheiten, http://www.automation.siemens.com/bilddb/download.asp?reqInsID=534843

5.2 Anhang B: Quellenangaben aus dem Internet

Autor unbekannt (2009), About OPC - What is OPC?, www.opcfoundation.org

Autor unbekannt (2009), CREATE TRIGGER, dev.mysql.com

Autor unbekannt (2009), Die Basissprachen KOP, FUP, AWL, support.automation.siemens.com

Autor unbekannt (2009), Feldbusse im Vergleich, www.feldbusse.de

Autor unbekannt (2009), Introduction to Ajax, www.prototypejs.org

Autor unbekannt (2009), OPC Batch Specification, www.opcfoundation.org

Autor unbekannt (2009), OPC Data Access Specification, www.opcfoundation.org

Autor unbekannt (2009), OPC Unified Architecture, www.opcfoundation.org

Autor unbekannt (2009), OPC XML-DA Specification, www.opcfoundation.org

Autor unbekannt (2009): OPCEx Excel Add-In, www.resolvica.com

Autor unbekannt (2009), The MySQL Native Driver, at2.php.net

5.3 Anhang C: Abbildungsverzeichnis

Abbildungsverzeichnis

Abbildung 1: Verschiedene Ausführungen von Bedienterminals (von links nach rechts): Textbasiert (Siemens AG 2009a: Text Display TD 400C), Mobil inkl. Touchscreen (Siemens AG 2009b: Mobile Panel 277F IWLAN) und Grafik inkl. Touchscreen (Siemens AG 2009c: THIN CLIENT PRO 15 Zoll mit Erweiterungseinheiten).....2

Abbildung 2: Linux Feldbus-Controller (WAGO 2008, S. 1) inkl. Abmessungen (WAGO 2006, S. 19).....18

Abbildung 3: Web-based Management: Information.....27

Abbildung 4: Web-based Management: I/O-Busklemmen.....28

Abbildung 5: Beispiel der Adressierung im Prozessabbild des Feldbus-Controllers (WAGO 2006, S. 1).....29

Abbildung 6: Gesamtkonzept für ein Summenstörmeldesystem (F: Fehler- bzw. Störmeldung).....36

Abbildung 7: Beispielbeschaltung des Feldbus-Controllers für ein Summenstörmeldesystem.....38

Abbildung 8: Übersicht des Testaufbaus für das Summenstörmeldesystem.....39

Abbildung 9: Der Testaufbau montiert auf einer Hutschiene bestehend aus Netzgerät und Feldbus-Controller mit Busklemmen.....40

Abbildung 10: Anschlüsse des Feldbus-Controllers (WAGO 2008, S. 1).....41

Abbildung 11: Busklemme 750-402 mit 4 digitalen Eingängen (WAGO 2009, S. 9).....43

Abbildung 12: Busklemme 750-504 mit 4 digitalen Ausgängen (WAGO 2009a, S. 11).....44

Abbildung 13: Busklemme 750-459 mit 4 analogen Eingängen (WAGO 2009b, S. 7).....46

Abbildung 14: Ansicht des Webbrowsers mit der Summenstörmeldeseite.....49

5.4 Anhang D: Tabellenverzeichnis

Tabellenverzeichnis

Tabelle 1: Systemdaten des Feldbus-Controllers 750-860
(vgl. WAGO 2008, S. 1).....42

Tabelle 2: Technische Daten des Feldbus-Controllers 750-860
(vgl. WAGO 2008, S. 2).....42

Tabelle 3: Technische Daten der Busklemme 750-402 mit 4 digitalen Eingängen (vgl. WAGO 2009, S. 12 f.).....44

Tabelle 4: Technische Daten der Busklemme 750-504 mit 4 digitalen Ausgängen (vgl. WAGO 2009a, S. 14 f.).....45

Tabelle 5: Technische Daten der Busklemme 750-459 mit 4 analogen Eingängen (vgl. WAGO 2009b, S. 9).....47

Tabelle 6: Prozessdaten der Busklemme 750-459 mit 4 analogen Eingängen
(* Statusbits: X = nicht benutzt, F = Kurzschluss, Ü = Überlauf)
(vgl. WAGO 2009b, S. 12).....47

5.5 Anhang E: Listingverzeichnis

Listingsverzeichnis

Listing 1: Terminalausgabe beim Systemstart des Testcontrollers: Kernelversion, CPU, RAM, Partitionen, Echtzeituhr, Journaling-Flash-FileSystem, Kbus-Treiber, angeschlossene Busklemmen.....21
Listing 2: Telnet-Login am Testcontroller mit Ausgabe der durch die Busybox zur Verfügung gestellten Befehle bzw. Programme.....22
Listing 3: Informationen zum Kbus-Treiber im /proc-Verzeichnis: aktuell am Feldbus-Controller angeschlossene Busklemmen inkl. Offsets im Eingangs- und Ausgangsprozessabbild.....23
Listing 4: Beim Systemstart wird die Datei "rc" im /etc-Verzeichnis aufgerufen. Diese ruft unter anderem auch die Feldbus-Controller-spezifische Datei "startwago" auf.....24
Listing 5: In der Datei "startwago" werden wichtige Dienste für den Feldbus-Controller gestartet: Kbus-Treiber, Modbus TCP (ein Feldbus), BOA Webserver und Terminal für die serielle Schnittstelle in Abhängigkeit des Schiebeschalters.....25
Listing 6: Nach dem Systemstart des Testcontrollers läuft der Kbus-Thread bzw. Kbus-Treiber im Hintergrund, welcher durch die Datei "startwago" aufgerufen wird.....26
Listing 7: Die am Testcontroller verfügbaren Wago-Treiber.....26
Listing 8: Das CGI-Verzeichnis für den BOA-Webserver mit dem CGI-Programm inkl. Links für das Wago-Web-based Management.....26
Listing 9: kbusapi.h: Die Header Datei der Kbus-API.....30
Listing 10: kbusapi.c: Der Quellcode der Kbus-API, 1. Teil.....31
Listing 11: kbusapi.c: Der Quellcode der Kbus-API, 2. Teil.....32
Listing 12: Mit dem Beispiel-Programm "kbusdemo" können nicht nur die Eingänge der Busklemmen Byte-weise ausgelesen werden, sondern auch die Ausgänge der Busklemmen Bit-weise gesetzt werden.....33
Listing 13: Auslesen des 13. Bytes aus dem Eingangs-Prozessabbild. Zuerst werden die einzelnen Bits dargestellt, dann als Hexadezimalzahl.....34
Listing 14: Setzen der Bits 12.1 und 12.0 im Ausgangs-Prozessabbild. In der Ausgabe erscheint das entsprechende Byte als Integerzahl bevor und nachdem das Prozessabbild geändert wurde.....34
Listing 15: index.html: Einbinden des Prototype Frameworks.....50

Listing 16: index.html: Eventhandler des Prototype Frameworks setzen.....50
Listing 17: index.html: Manuelles Laden der Daten vom Webserver.....51
Listing 18: index.html: Quittieren aktiver Störmeldungen.....51
Listing 19: index.html: Anfrage an Webserver nach Änderungen.....52
Listing 20: index.html: Anzeige des animierten „Loading ...“ Textes.....53
Listing 21: index.html: „Body“ mit „div“ in das die Störmeldeliste eingefügt wird.....53
Listing 22: "ssm"-DB: "messages"-Table.....54
Listing 23: "ssm"-DB: "lastChange"-Table.....54
Listing 24: "ssm"-DB: "renewTimeStamp"-Trigger.....55
Listing 25: Webserver: loadMessages.php.....56
Listing 26: Webserver: updateMessages.php.....57
Listing 27: Webserver: quittMessages.php.....58
Listing 28: Webserver: updateDI.php.....59
Listing 29: C-Programm am Feldbus-Controller, Teil 1 von 4.....61
Listing 30: C-Programm am Feldbus-Controller, Teil 2 von 4.....62
Listing 31: C-Programm am Feldbus-Controller, Teil 3 von 4.....63
Listing 32: C-Programm am Feldbus-Controller, Teil 4 von 4.....64

5.6 Anhang F: Danksagungen

Vielen Dank an meinen Betreuer Hrn. DI Grischa Schmiedl für die tatkräftige und geduldige Unterstützung bei der Ausarbeitung meiner Diplomarbeit.

Ein Danke auch an die Firmen IGEA und WAGO, die einen Aufbau des Projekts erst durch ihren Verleih des benötigten Materials ermöglicht haben.