Fachhochschule
Pforzheim

Hochschule
für Gestaltung
Technik und
Wirtschaft

Pforzheim University
of Applied Sciences

SCADA-EASy

Alexander Thiel

In vielen Unternehmen gibt es eine inhomogene Systemlandschaft mit mehr oder weniger autarken Automationszellen. In diesen fallen eine Menge von Daten und Informationen an, welche an zentraler Stelle erfasst, zur Optimierung der einzelnen Prozessabläufe beitragen können. Die Erfassung und Visualisierung dieser Daten ist Aufgabe des hier beschriebenen Ethernetbasierten Automatisierungssystems.

Fachhochschule Pforzheim
Fachbereich Automatisierungstechnik

Interdisziplinäre Projektarbeit

SCADA-EASy

Alexander Thiel
Mat-Nr. 285800

18. August 2002

Die Projektarbeit wurde betreut von:
Prof. Dr. Frank Thuselt
Prof. Dr. Schätter
Dipl. phys. Michael Bauer

Inhalt

1 Das Projekt

1.1 Projektbeschreibung

In vielen Unternehmen gibt es eine inhomogene Systemlandschaft mit mehr oder weniger autarken Automationszellen. In diesen fallen eine Menge von Daten und Informationen an, welche an zentraler Stelle erfasst, zur Optimierung der einzelnen Prozessabläufe beitragen können. Die Erfassung und Visualisierung dieser Daten ist Aufgabe des hier beschriebenen Ethernetbasierten Automatisierungssystems.

1.2 Vorgehensweise

1. Im Rahmen der Projektarbeit ist ein UNIX[1] basierter Server aufzubauen und zu konfigurieren, der mit verschiedenen Clients unterschiedlichster Plattformen kommunizieren kann.
2. Der Server soll datenbankbasiert Informationen erfassen und dynamisch den Bedürfnissen der verschiedenen Anwender entsprechend die Daten zur Verfügung stellen.
3. Als Beispiel für eine Prozessstation soll ein Client auf Microcontroller Basis mit integrierter Messdatenerfassung (z.B. Temperaturmessung) aufgebaut werden, welcher über das vorhandene LAN dem Server die Daten übergibt.

1.3 Ziele

Die im folgenden gezeigten Verfahren und Vorgehensweisen sollen nicht die breite Palette proprietärer Systeme zur Wartung und Analyse von Daten aus Automationszellen um ein neues Produkt erweitern. Vielmehr soll die vorliegende Arbeit als Grundlage einer unabhängigen Beschreibungssprache für die Visualisierung von Prozessdaten dienen.

[1] In unserem Fall handelt es sich um einen Intel-PC mit einer SuSe Linux 7.2 Installation

Hierbei steht in der ersten Ausbaustufe die Abbildung der logischen Strukturen der Automationszellen im Vordergrund. Die Darstellung der aufgenommenen Daten erfolgt hier noch mit festgelegten Programmen (den Viewer-Applets). Aber ein offenes Projekt wäre kein solches, wenn nicht auch die Möglichkeit zur Beschreibung dieser Darstellungen in Betracht kommen könnte.

Auf der Basis einer solchen Beschreibungssprache lässt sich das Projekt unabhängig von unserem Beispiel in vielfältiger Weise überall dort einsetzen wo Messwerte ermittelt und aufbereitet werden müssen.

Sei es in der Automatisierungstechnik zur Visualisierung von Prozessabläufen oder um die Daten einer Wetterstation im Internet darzustellen. All diese Projekte können auf dem SCADA-EASy System aufgebaut werden.

Aufgrund dieser Möglichkeiten hoffe ich, dass die vorliegende Arbeit als Grundlage weiterer Entwicklungen im Bereich der Prozessdatenvisualisierung dient und in diesem Rahmen auch verbessert und weiter ausgebaut wird.

2 Hardware und Systemsoftware

An dieser Stelle soll die Installation unseres Beispielservers kurz erläutert werden. Diese Beschreibung ersetzt nicht das Systemhandbuch oder ähnliche Dokumentation der Systemsoftware. Sie stellt auch keine Einführung in Linux dar.

2.1 Vorüberlegungen

Als Betriebssystem für unseren Server haben wir SuSe Linux in der Version 7.2 vorgesehen. Der Rechner ist mit einem SCSI-System ausgerüstet und hat zur Erweiterung eine IDE-Festplatte mit Wechselrahmen, welche zur Datenhaltung vorgesehen ist. In früheren Projekten hat sich gezeigt, dass Rechnersysteme mit IDE und SCSI Technologie Probleme bereiten. YAST[1] installiert den Masterbootrekord[2] (MBR) auf der IDE Platte, da dieses Installationstool davon ausgeht, dass Systeme, welche IDE Technologie nutzen, dies ausschliesslich tun. Im Falle eines Ausbaus unserer Wechselplatte wäre unser System daher nicht mehr bootfähig. Um dieses Problem zu umgehen installieren wir das System bei ausgebauter IDE-Festplatte. Nun wird der MBR wunschgemäss auf der SCSI-Festplatte installiert, und unser System bleibt auch bei ausgebauter Wechselplatte weiterhin lauffähig.

Zu Beachten ist allerdings, dass bei Nutzung des YAST Installationstools immer die Wechselplatte ausgebaut ist. Dies trifft zwar nicht für die Installation von zusätzlichen Softwarepaketen zu, jedoch wird beim kompilieren des Systemkernels oder einem Systemupdate der MBR neu beschrieben und unser System damit zerstört.

2.2 Installation

[1] Das Installationstool der SuSe Distribution

[2] Der physikalisch erste Sektor der Festplatte im System. Hieraus lädt das BIOS den Code welcher das Betriebssystem startet.

2.2.1 Systemzeit

Im Prinzip kann man die Standardinstallation des YAST Installationstools verwenden. Doch sollte die Vorgabe der Systemzeit von *Ortszeit* auf *GMT*[3] umgestellt werden, da dies für Server üblich ist, welche im Internet betrieben werden, um einen weltweit einheitlichen Zeitstandard gewährleisten zu können. Auch wir müssen hierbei bedenken, dass der Server mit Daten arbeiten muss, deren geographische Herkunft nicht bekannt ist. Daher die Wahl der GMT.

2.2.2 Partitionierung

Nach der Einstellung der Systemzeit wird man aufgefordert, eine Partition für die Installation auszuwählen. An diesem Punkt sollte nicht vergessen werden, dass Linux als *virtuellen Speicher* eine *swap* Partition benötigt. Wir legen daher eine Partition mit 64MB für diesen Zweck an. Eine Grösse von 64MB oder 128MB sind für heutige Systeme ausreichend. Details zum Verfahren der Grössenberechnung finden Sie in [MS98]. Die Grösse dieser Partition ergibt sich aus der Grösse des Hauptspeichers des Servers.

2.2.3 Das System

Wir kommen nun an den Punkt, an dem wir aufgefordert werden, einen Installationstyp zu wählen. Folgende Typen stehen zur Auswahl:

1. minimales System
2. minimales graphisches System (ohne KDE)
3. Standard-System
4. Standard-System mit Office
5. Alle Pakete

Für einen reinen Server wäre das *minimale System* ohne weiteres ausreichend. Da wir aber davon ausgehen, dass die Administration und weitere Nutzung des Systems nicht immer von geübten UNIX Anwendern vorgenommen wird, spendieren wir dem System eine minimale X-Windows Umgebung indem wir das *minimale graphische System* installieren.

[3] Greenwich Meantime

Der Einsatz von KDE, wie es in den Paketen mit *Standard-Systemen* der Fall wäre, ist für unseren kleinen Server viel zu resourcenhungrig, sodass eine solche Installation unser System nur unnötig belasten würde.

2.2.4 Netzwerkkonfiguration

Da der Server während der Installation nicht in seinem endgültigen Netzwerk integriert war, werden wir an dieser Stelle erläutern, wie die Netzwerkeinstellungen zur Anpassung an den endgültigen Standort vorzunehmen ist.

Um die Netzwerkadressen einzustellen, verwenden wir das Konfigurationstool YAST2 von SuSe. Beachten Sie jedoch, dass dieses Tool nur die Grundeinstellungen in Ihrer `rc.config`-Datei anpasst. Änderungen, welche von Hand in Einstellungsdateien von Netzwerkdaemons[4] vorgenommen wurden, werden dadurch nicht beeinflusst und müssen dementsprechen von Hand wieder korrigiert werden.

2.2.5 Konfiguration des X-Servers

An dieser Stelle muss ich leider ein kleines Missgeschick offenbaren. Leider haben meine hastigen Finger den Installationsprozess vor Einrichtung des X-Servers abgebrochen und unser Server startet nun im Textmodus. Ich kann an dieser Stelle also nicht die Konfiguration mittels YAST beschreiben, sondern werde ausführen wie die Installation und Konfiguration mittels SAX nachträglich vorzunehmen ist.

Melden Sie sich an der Textkonsole als *root* an und starten Sie durch Eingabe von `sax2` das SAX Konfigurationstool. Dieses Tool ist graphisch geführt und weitestgehend selbsterklärend, sodass wir von einer detaillierteren Anleitung an dieser Stelle absehen. Es sei allerdings noch gesagt, dass Sie sich eingehend mit der vorhandenen Hardware beschäftigen sollten, bevor Sie den X-Server konfigurieren. Kenntnisse über die Ablenkfrequenz Ihres Monitors und detaillierte Informationen über Ihre Grafikkarte sind unbedingt nötig. Sollten Sie falsche Angaben zu Ihrer Hardware machen kann der Monitor nichtreparable Schäden davontragen.

Nach der Konfiguration können Sie nun die graphische Oberfläche X-Windows nutzen. Melden Sie sich am System an und starten Sie den X-Server durch Eingabe von `startx`. Nun wird die graphische Oberfläche gestartet. Um den Umgang mit dieser Oberfläche aber noch ein wenig komfortabler zu gestalten, können wir auch schon die Anmeldung als graphische Oberfläche einrichten. Dies geschieht mit Hilfe der XDM Technologie.

[4] Programme, welche Systemdienste zur Verfügung stellen, z.B. der Apache Webserver oder der *inetd* Superdaemon.

Das graphische Login können Sie mit dem Tool `yast` unter der Rubrik *Loginkonfiguration* einrichten, indem Sie als Loginmodus *graphisch* wählen.

2.2.6 Einrichtung der Datenplatte

Wie schon geschildert soll eine IDE-Wechselfestplatte zur Datenhaltung dienen. Diese Festplatte muss nun im System noch konfiguriert und eingerichtet werden.

Als Erstes muss dazu die Festplatte partitioniert (in unserem Fall nur eine primäre Partition) und formatiert werden. Zur Partitionierung geben Sie an der Textconsole (oder in einem Terminal der graphischen Benutzeroberfläche) `fdisk /dev/hda` ein. Hierbei steht `/dev/hda` für die erste IDE-Festplatte. Die Nutzung dieses Tools ist sehr gut in der Hilfe beschrieben, welche durch Eingabe von `m` aufgerufen werden kann. Legen Sie nun so viele Partitionen wie gewünscht an und formatieren Sie diese durch Eingabe von `mk2fs /dev/hda1`, wobei die Zahl am Ende der Plattenbezeichnung die Nummer der Partition angibt.

Wir wollen nun die neue Festplatte in unser Dateisystem einbinden. Da auf dieser Platte die WWW-Dateien abgelegt werden sollen und der Zugriff darauf sehr einfach gestaltet werden soll (ist ja auch ein Webserver), legen wir ein neues Verzeichnis durch Eingabe von `mkdir www` an. Dies geschieht alles auf der root-Ebene (`/`) und wir erzeugen dadurch das Verzeichnis `/www`. Dieses Verzeichnis hat nun aber nichts mit unserer Festplatte zu tun, denn diese müssen wir erst hierher mounten. Geben Sie einfach `mount /dev/hda1 /www` ein und die Festplatte wird als dieses Verzeichnis gemountet.

Bei einem Neustart allerdings geht dieser `mount`-Befehl wieder verloren und wir müssten die Festplatte neu in das Verzeichnis mounten. Um dies zu vermeiden, schreiben wir die Festplatte in unsere `fstab` Datei, welche beim Booten des Systems aufgerufen wird und zusätzliche Laufwerke mountet. Unser Eintrag sieht folgendermassen aus:

```
/dev/hda1 /www ext2 defaults 1 1
```

Damit sollte die Festplatte auch nach einem Neustart ordentlich gemountet sein.

3 Funktionsweise

In diesem Kapitel möchte ich Ihnen die Struktur des SCADA-EASy Systems und damit die Idee, welche dahinter steht, näher bringen. Auf die Codierung des Systems soll aber nicht eingegangen werden. Hier verweise ich auf die kommentierten Quellcodes.

3.1 Grundstruktur

SCADA-EASy dient als Kommunikationseinheit zwischen Messwertaufnehmern und dem Anwender. Die Messwertaufnehmer (im folgenden *Prozessstationen* genannt) senden ihre Messdaten an den Prozessserver, welcher diese aufbereitet und in eine Datenbank[1] ablegt. Der Anwender hat nun die Möglichkeit, mit Hilfe von speziellen Viewer-Applets[2], die Daten am Browser anzeigen zu lassen. Das in Abbildung 3.1 dargestellte Diagramm soll diese Verbindungen veranschaulichen. Die in Abbildung 3.1 dargestellten Pfeile sym-

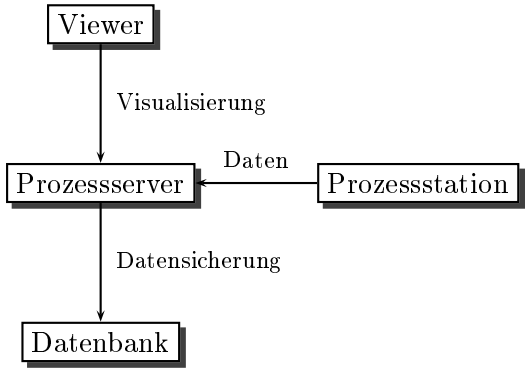

Abbildung 3.1: Kommunikationsschema des Automatisierungsportals

bolisieren die Initiationsrichtung der Kommunikation. Es fällt auf, dass der Server selbst

[1] Der Begriff Datenbank ist nicht ganz korrekt, da die Daten nicht in Form einer klassischen Datenbank gehalten werden. Der Einfachheit halber wurde das System jedoch hier so bezeichnet.

[2] Die Bezeichnung Applet wurde in Anlehnung an Java-Applets gewählt, da die Visualisierung auf Clientseite auf dieser Technologie basiert.

nur zur Datenbank eigenständig die Kommunikation aufnimmt. Prozessstationen und Viewer müssen nicht auf Serveranfragen reagieren, sie holen ihre Daten vom Server ab. Das beinhaltet aber, dass die einzelnen Teilnehmer (Prozessstationen und Viewer) einen entsprechenden Algorithmus für das Datenpolling implementiert haben. Im einfachsten Fall geschieht dies zyklisch mit Hilfe eines Timers. Nachteil dieses Verfahrens ist aber, dass keine ereignisgesteuerten Prozesse ablaufen können und die Verzögerungszeit recht hoch sein kann.

Auf Basis des Parameters TIM, welcher den Refreshtimer für Viewer und Modul im Sekundentakt festlegt, ergibt sich für die Verzögerungszeit $t_{max} \leq \sum TIM_n$. Da die Verzögerung TIM einmal vom Modul zum Portal und ein zweites Mal vom Portal zum Viewer berücksichtigt werden muss, erhalten wir im ungünstigsten Fall bei einer Refreshrate von $TIM = 1s$ eine maximale Verzögerung von $t_{max} \leq 2 \bullet TIM \Rightarrow t_{max} \leq 2s$. Da das System mit Sekundentakten arbeitet, ist der Wert von $2s$ als Standardverzögerung vorgegeben. Systeme, welche eine niedrigere Verzögerung benötigen, sollten nicht mit dem momentan verfügbaren System oder nur mit äusserster Vorsicht betrieben werden.

Aus den geführten Betrachtungen ergibt sich, dass das Portal für Langzeiterfassungen sehr gut geeignet, für zeitkritische Anwendungen hingegen ungeeignet ist. Dies wird durch die *Flankenempfindlichkeit* der Datenbank unterstützt. Das bedeutet, dass die Datenbank nur auf Änderungen von Werten reagiert und nur den Zeitpunkt der Änderung speichert. Hierdurch kann das Datenvolumen bei trägen Systemen enorm reduziert werden. Dies trifft insbesondere dann zu, wenn der Datentransfer zyklisch erfolgt und nicht mittels eines Δ-Algorithmus[3] im Modul.

Um die Kommunikation der einzelnen Teilnehmer mit dem Portalserver zu erläutern werde ich die verwendeten Protokolle für jede Verbindung separat beschreiben, da diese abhängig von der Art des Teilnehmers (Prozessstation oder Viewer) sind.

3.2 Prozessstation-Server-Kommunikation

Prozessstationen senden an den Server ihre aktuellen Zustandsdaten und erhalten von diesem auf Anfrage Sollwertvorgaben, welche ein Beobachter mittels Viewer auf dem Server abgelegt hat. Hierzu wird immer nur ein Datensatz versendet oder empfangen, da eine Prozessstation immer nur den *Status Quo* beschreibt oder liest. Es wird vom Kommunikationsprotokoll auch keine Identifizierung der Prozessstation benötigt, da diese der IP-Adresse der Prozessstation entspricht und aus dem TCP/IP Protokoll direkt ausgelesen wird. Dieses Verfahren gewährleistet immer eine eindeutige Identifizierung der Prozessstation, beinhaltet aber auch gewisse Einschränkungen. So ist es zum Bei-

[3] Algorithmus, welcher Zustandsänderungen der Eingangsvariablen erkennt.

spiel nicht möglich mehrere Prozessstationen über ein Gateway mit NAT[4] zu betreiben, da dieses Gateway die IP-Adresse der Prozessstation mit der Eigenen überschreibt und eine eindeutige Identifizierung verschiedener Prozessstationen hinter einem solchen Gateway nicht mehr möglich ist. Die Prozessstationen sollten im besten Fall eine direkte Netzwerkverbindung zum Server aufweisen.

Das von uns verwendete Protokoll basiert auf dem Common Gateway Interface (CGI), auf Basis von HTML[5]. Diese Schnittstelle stellt die Basis aller Kommunikationsprozesse mit dem Server dar. Vorteil dieses Vorgehens ist die einfache Implementation unter verschiedenen Webservern, da diese alle dieses Protokoll unterstützen und dadurch keine propriätäre Verbindungen notwendig sind. Es ist aber auch möglich andere Protokolle zu verwenden. So haben wir für Testzwecke auch ein CLI (Command Line Interface) entwickelt. Das verwendete Gateway wird durch das Package `gateway.pm` spezifiziert; dazu aber später mehr. Unser CGI Package verarbeitet sowohl die `GET` als auch die `PUT` Methode von CGI. Beide Varianten werden unterstützt, da sich je nach Anwendungsfall verschiedene Anforderungen ergeben. So lässt sich die `GET` Methode recht einfach implementieren und bildet auch die Basis unseres Demo-Controllers. Die `PUT` Methode verbirgt die gesendeten Daten besser und eignet sich daher auch für grössere Datenmengen wie Binärdateien[6].

Von den Prozessstationen werden also zwei Befehle an den Server abgesetzt. Zum einen ein `write` Kommando zum Schreiben der aktuellen Einstellungen auf den Server, zum anderen ein `read` Kommando zum Auslesen der Sollwerte. Da die Identifizierung der Prozessstation, wie schon erwähnt, über das TCP/IP Protokoll erfolgt, benötigt das Kommando `read` keine weiteren Parameter und kann in der Form

```
http://server-ip/cgi-bin/modules.cgi?cmd=read
```

zum Server geschickt werden. Dieser generiert nun einen Textoutput, welcher an die Prozessstation zurückgegeben wrd. Dieser Textoutput sieht in etwa so aus:

```
BOT&D00=23&D01=11&A00=12.45&EOT
```

Die einzelnen Teile dieser Nachricht sind durch ein *Ampersand* getrennt und enthalten die Sollwerte für die einzelnen Prozessdateneinheiten (PDUs). In unserem Beispiel für PDU `D00` den Wert `23`, für PDU `D01` den Wert `11` usw. Die beiden Teile vor und am Ende der eigentlichen Nachricht spezifizieren *Begin of Transmission* (`BOT`) und *End of Transmission* (`EOT`). Diese beiden Werte werden für das Protokoll nicht direkt benutzt,

[4] Network Address Transformation
[5] Hypertext Markup Language. Seitenbeschreibungssprache des Internets.
[6] Zur Zeit werden Binärdaten noch nicht vom Server unterstützt.

wurden aber eingeführt, um die Erkennung einer Parameterübertragung zu kennzeichnen, da jeder Befehl eine Rückmeldung vom Server erzeugt, und um so diese wichtige Parameterübertragung von eher unwichtigen Bestätigungsmeldungen zu unterscheiden.

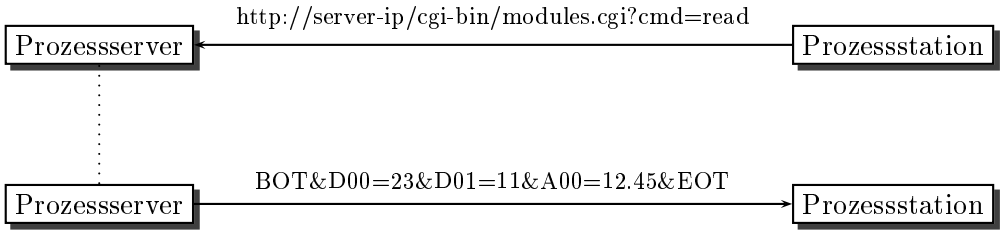

Abbildung 3.2: Lesezugriff einer Prozessstation auf den Server

In den gezeigten Beispielen fällt auf, dass jede Datenübertragung mit Hilfe von Wertepaaren[7] erfolgt. Listen von Wertepaaren werden in der Softwareentwicklung auch als *assoziative Listen*, *assoziative Arrays* oder *Hash* bezeichnet. In der folgenden Dokumentation wird der Begriff *assoziatives Array* verwendet. Näheres zu diesem Thema finden Sie auch in [Sch00]. Ein solches *assoziatives Array* zeichnet sich durch die Zuordnung eines Wertes zu einem eindeutigen Schlüssel aus. Die Schlüssel (Keys) für die Prozessstation-Server-Kommunikation sind in Tabelle 3.1 zusammengestellt. Für

Tabelle 3.1: Schlüssel der Prozessstation-Server-Kommunikation

Key	**Funktion**
CMD	Legt das auszuführende Kommando fest. Dies wird benötigt, da sonst kein Befehl ausgeführt wird!
TIM	Gibt die Zykluszeit für Schreibzugriffe der Prozessstation auf den Server an. Ist hier nichts spezifiziert, obliegt der Zyklus dem Algorithmus der Prozessstation. (Für Prozessstationen nur lesbar!)
A00..A99	Prozessdateneinheit (PDU) für analoge Werte. Es stehen maximal 100 analoge PDUs zur Verfügung.
D00..D99	Prozessdateneinheit für digitale Werte. Die Daten der PDUs werden mittels eines 8 bit Integerwertes in Dezimaldarstellung übertragen, z.B. D00=129

sie ist eine feste Länge von 3 Zeichen vorgesehen. Dies wurde für die Station-Server-

[7] Ausgenommen sind hier BOT und EOT, welche nicht direkt zum Protokoll gehören

Kommunikation in dieser Form festgelegt, um die Verarbeitung auf den Prozessstationen, welche oftmals aus einfachen Microrechnern bestehen, zu vereinfachen.

3.3 Viewer-Server-Kommunikation

Umfangreicher wird die Kommunikation der Viewer mit dem Server. Hier muss gewährleistet werden, dass der Viewer auf eine bestimmte Prozessstation zugreift, somit muss er auch deren IP-Adresse mitliefern. Da Viewer in der Regel nur eine Prozessdateneinheit visualisieren[8], muss auch die Kennung des entsprechenden Ports an den Server geliefert werden. Weiterhin unterstützt der Server verschiedene Antwortarten, was den Befehlsumfang erweitert. Abbildung 3.3 zeigt das Kommunikationsschema. Die *Anforderung*

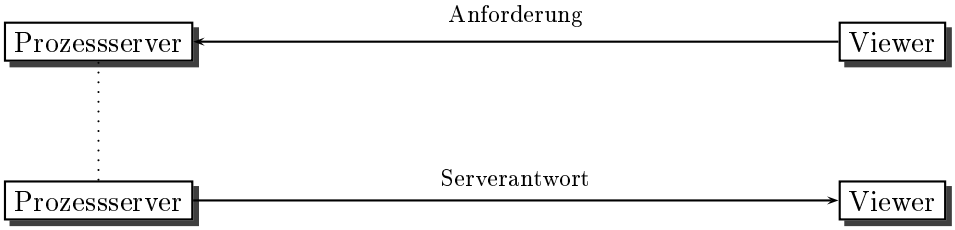

Abbildung 3.3: Lesezugriff eines Viewers auf den Server

könnte zum Beispiel folgendermassen Aussehen:

```
http://server-ip/portal.cgi?ip=141.47.75.18&pdu=D00&cmd=getLast
```

Die Antwort vom Server würde daraufhin wie folgt lauten:

```
#Data from PDU D00 on PST 141.47.75.18
Sun May  5 22:56:09 2002 53
```

Wie in diesem Beispiel zu sehen, ist das Protokoll zwischen Viewer und Server umfangreicher als das zwischen Prozessstation und Server. So sind die Schlüssel nicht mehr auf 3 Zeichen beschränkt. Auch das Antwortfile vom Server enthält nun mehr Informationen. Dieses kann auch mehrzeilig ausfallen, wenn das Kommando `getAll` verwendet wird. Es liefert dann alle bisher aufgezeichneten Werte zurück.

[8] Es ist auch möglich, Viewer für mehrere PDUs zu schreiben, aber in der aktuellen Version in dieser Form nicht vorgesehen.

```
#Data from PDU D00 on PST 141.47.75.18
Sun May  5 20:33:30 2002 23
Sun May  5 20:38:29 2002 21
Sun May  5 21:04:06 2002 23
Sun May  5 21:06:12 2002 233
Sun May  5 21:13:35 2002 23
Sun May  5 21:23:23 2002 233
Sun May  5 21:29:10 2002 23
```

Die Anzahl der möglichen Schlüssel verringert sich aber, da ein gleichzeitiges beschreiben mehrerer Prozessdateneinheiten wie Beispielsweise beim `write`-Kommando der Prozessstation-Server-Kommunikation, nicht mehr vorgesehen ist. Die bisher verfügba-

Tabelle 3.2: Schlüssel der Viewer-Server-Kommunikation

Key	**Funktion**
CMD	Legt das auszuführende Kommando fest. Dies wird benötigt, da sonst kein Befehl ausgeführt wird!
IP	Spezifiziert die gewählte Prozessstation.
PDU	ID der gewählten Prozessdateneinheit.
VALUE	Wird benötigt, um Werte an die Prozessdateneinheit zu übergeben.

ren Kommandos für den `CMD`-Parameter sind in Tabelle 3.3 aufgeführt.

Tabelle 3.3: Kommandos für Viewer

Kommando	**Funktion**
getLast	Gibt den aktuellen Zustandswert zurück.
getAll	Gibt alle gespeicherten Werte in chronologischer Reihenfolge zurück.
getCSV	Gibt alle gespeicherten Werte in chronologischer Reihenfolge in Form eines *tabular-separated-file* zurück.
Module	Gibt eine HTML-Seite mit allen zu dieser Prozessstation gehördenden Informationen aus.
Port	Gibt eine HTML-Seite mit dem Viewer-Applet für diese Prozessdateneinheit zurück.
setValue	Schreibt den mit VALUE spezifizierten Wert als Sollwert fest.

3.4 Beschreibungsdateien

In diesen Dateien wird der logische Aufbau der Prozessstationen und der in diesen enthaltenen Prozessdateneinheiten beschrieben. Hierbei spielt aber der physikalische Aufbau des Systems keine Rolle, allein die logische Struktur soll wiedergegeben werden. Um dies zu realisieren wurde als Basis für die Beschreibungsdateien XML[9] als Auszeichnungssprache gewählt. Eine detailliertere Beschreibung von XML kann im Rahmen dieser Arbeit nicht erfolgen. Die grundlegenden Konzepte können aber in [Mic99] nachgeschlagen werden.

3.4.1 Struktureller Aufbau

Als Wurzelelement könnte dem Automatisierungsportal eigentlich die Automationszelle dienen. Hierbei würde pro Automationszelle nur eine Datei angelegt. Da die Struktur eines solchen Systems aber durchaus einem gewissen Wandel unterlegen ist, wäre in diesem Fall die Pflege sehr hoch, da beim Abschalten einer PDU die komplette Automationszelle verändert werden müsste.

Sollten Beschreibungsdateien von Herstellern der Prozessstationen in das System integriert werden, dann müssten diese vollständig in die Beschreibung der Automationszelle einfliessen. Die Fehleranfälligkeit eines solchen Verfahrens wäre jedoch sehr hoch, daher wird im vorliegenden System eine Automationszelle durch einen Ordner mit den Beschreibungsdateien der einzelnen Prozesstationen repräsentiert.

Der Dateiname der Beschreibungsdatei beinhaltet die IP-Adresse der Prozessstation, um eine eindeutige Zuordnung der Beschreibungsdateien zu den Prozessstationen zu ermöglichen. Der Name einer solchen Datei hat demnach die Form `nnn.nnn.nnn.nnn.xml`.

Innerhalb einer Beschreibungsdatei bildet die Prozessstation selbst das Wurzelelement. Die Prozessdateneinheiten werden diesem als Childelemente zugeordnet.

3.4.2 Dokumenttyp-Definitionen (DTD)

Um Inhalte in XML-Instanzen (also Dokumente, die einer DTD entsprechen) auszeichnen zu können, müssen die in dieser Instanz enthaltenen Elemente in einer DTD deklariert werden. DTDs sind also Sammlungen von *Deklarationen*. Nur was in der DTD deklariert wurde, darf in der XML-Instanz verwendet werden.

[9] Extensible Markup Language

Die hier erläuterte Deklaration ist demzufolge das *Herzstück* des Beschreibungssystems. Anhand eines Beispieles lässt sich der Aufbau einer solchen Beschreibungsdatei recht einfach erkennen.

```
<PST>
   <NAME>Teststation</NAME>
   <REFRESH>10</REFRESH>
   <PDU id="D00">
      <NAME>Erster Digitalport</NAME>
      <VIEWER type="simpleview">
         <UDD name="backgroundcolor">#FFFFFF</UDD>
         <UDD name="foregroundcolor">#000000</UDD>
      </VIEWER>
   </PDU>
   <PDU id="D01">
      <NAME>Zweiter Digitalport (Ausgang)</NAME>
      <WRITABLE>true</WRITABLE>
      <VIEWER type="simpleview">
         <UDD name="backgroundcolor">#FFFFFF</UDD>
         <UDD name="foregroundcolor">#000000</UDD>
      </VIEWER>
   </PDU>
   <PDU id="A00">
      <NAME>Erster Analogport (Temperatursensor)</NAME>
      <VIEWER type="simpleview">
         <UDD name="backgroundcolor">#FFFFFF</UDD>
         <UDD name="foregroundcolor">#000000</UDD>
      </VIEWER>
   </PDU>
   <PDU id="A01">
      <NAME>Zweiter Analogport</NAME>
      <VIEWER type="simpleview">
         <UDD name="backgroundcolor">#FFFFFF</UDD>
         <UDD name="foregroundcolor">#000000</UDD>
      </VIEWER>
   </PDU>
</PST>
```

In dieser Datei wird eine Prozessstation mit 4 Prozessdateneinheiten (2 analoge und 2 digitale Einheiten) beschrieben. Das Wurzelelement stellt die `<PST>` dar. Als weitere

Elemente können hier der Name[10] und die Refreshtime[11] angegeben werden.

Jede Prozessdateneinheit bildet ein Subelement der `<PST>` und wird mit `<PDU>` bezeichnet. Jedes `<PDU>`-Element muss mit einer eindeutigen ID gekennzeichnet sein. Diese ID ist die schon im Kommunikationsprotokoll verwendete Kombination aus dem Typenkurzzeichen (A für analog, D für digital) und der laufenden Nummer der Prozessdateneinheit, also `D00` für die erste digitale PDU.

Auch einer PDU kann mittels `<NAME>`-Element eine Klartextbezeichnung zugeordnet werden. Ein weiteres Element, das `<VIEWER>`-Element, ist aber weitaus interessanter. Hier wird der zur Visualisierung der Daten eingesetzte Viewer spezifiziert. Das Attribut `type` gibt die Art[12] des zu verwendenden Viewers an. In dem gezeigten Beispiel wird der Default-Viewer *simpleview* in allen 4 PDUs verwendet. Dieser Viewer müsste nicht explizit angegeben werden, da bei fehlender Viewerangabe dieser automatisch verwendet wird.

Die Subelemente `<UDD>` (**U**ser **D**efined **D**ata) des Viewers werden verwendet, um Informationen, welche nicht in der Basisdefinition der Beschreibungssyntax enthalten sind, an den Viewer weiterzureichen. Die hier vergebenen Werte werden in der Form `backgroundcolor=#FFFFFF` an den Viewer weitergeleitet. Diese Vorgehensweise erlaubt es auch komplizierte Viewer mit Hilfe der Beschreibungsdatei zu konfigurieren. Der von uns verwandte Viewer *simpleview* würde die zusätzlichen Informationen ignorieren, da dieser keine Parameter erwartet.

Die *DTD* für das gezeigte Beschreibungsmodell sieht nun folgendermassen aus:

```
<!ELEMENT PST (NAME, REFRESH, PDU+)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT REFRESH (#PCDATA)>
<!ELEMENT PDU (NAME, WRITABLE, VIEWER)>
<!ATTLIST PDU id ID #REQUIRED>
<!ELEMENT WRITABLE (#PCDATA)>
<!ELEMENT VIEWER (UDD*)>
<!ATTLIST VIEWER type CDATA #REQUIRED>
<!ELEMENT UDD (#PCDATA)>
<!ATTLIST UDD name CDATA #REQUIRED>
```

Im vorliegenden Prototypen des Prozessservers werden die Beschreibungsdateien allerdings noch nicht nach diesen Regeln auf Validität geprüft. Zukünftige Weiterentwick-

[10] Eine Klartextbezeichnung, um die einzelnen Prozesstationen später auch namentlich und nicht nur mittels IDs benennen zu können.
[11] Zykluszeit für den Pollingbetrieb in Sekunden.
[12] Dies ist gleichzeitig auch der Dateiname des JAR-Archives des Viewers.

lungen werden, da die Beschreibung solcher Netzstrukturen sehr strengen Regeln unterliegt, wahrscheinlich auch hier die Restriktionen genauer prüfen. Der Einsatz von *XML-Schemata* zur Deklaration der Beschreibungssyntax wird zum Beispiel gerade geprüft. Hiermit liesse sich die Syntax noch genauer festlegen. Zur Fehlervermeidung wäre dann auch die Definition von Datentypen möglich.

3.5 Datenhaltung

Im aktuellen Prototypen werden die von den Prozessstationen empfangenen Daten auf Änderung geprüft. Dies ist der schon anfangs erwähnte *Delta*-Algorithmus. Wurde eine Änderung zum vorhergehenden Prozesswert festgestellt, wird der Datensatz mit einem Zeitstempel versehen und in einem Hash abgespeichert. Diese Vorgehensweise erzeugt so für jede Prozessdateneinheit eine eigene Datei, in welcher die Datenpaare (Zeitstempel, Wert) gehalten werden.

Zur Zeit werden diese Dateien in ein festgelegtes Verzeichnis auf dem Server abgelegt. Es wurden aber schon Überlegungen angestellt, den Ablageort dieser Daten in der Beschreibungsdatei zu definieren. Dies hätte den Vorteil, dass die Datenablage sowohl in besagter Hash-Struktur, als auch in einem Datenbanksystem möglich wäre. Dies wäre dann auch in einem gemischten Betrieb für jede Prozessdateneinheit separat in der Beschreibungsdatei spezifizierbar.

4 Implementierung

4.1 Hardware und Systemsoftware

Der für den Aufbau verwendete Server ist ein ausgemustertes *Pentium I* Gerät, welches mit Hilfe einer einfachen Linux Distribution wieder zu neuem Leben erweckt wurde. Wie im Kapitel *Hardware und Systemsoftware* beschrieben, wird für die Kommunikation mittels HTTP der recht weit verbreitete Apache-Webserver verwendet.

Weiterhin wird auf dem System eine Installation von Perl 5 verwendet. Die Gründe für den Einsatz von Perl werden im Folgenden noch detaillierter geschildert.

4.2 SCADA-EASy

Die Servertechnik wurde für unseren Prototypen vollständig in Perl codiert. Da die objektorientierte Vorgehensweise verwendet wird, wird Perl Version 5 oder höher vorausgesetzt. Die Wahl von Perl liegt in den typischen Eigenschaften einer Scriptsprache begründet. Diese Sprachen sind leicht zu erlernen und aufgrund ihrer Interpretierung sind Änderungen des Codes leicht durchführbar.

An der Wahl der Programmiersprache war der jedoch schon in die Jahre gekommene Server nicht ganz unbeteiligt. So wurde festgestellt, dass die Perl-Implementierung in einer recht akzeptablen Geschwindigkeit interpretiert wurde. Der Versuch, dies mit Hilfe eines *JAVA-Servlets* zu realisieren, scheiterte, da die *VirtualMachine* auf dem Rechner zu resourcenhungrig war.

In den vorangegangenen Kapiteln wurde die Struktur des Systems sehr detailliert geschildert. Wie teilweise bereits dort angemerkt, sind nicht alle Konzepte im Prototypen verwirklicht worden. Dies hängt vor allem mit der begrenzten Projektzeit zusammen, die gewisse Funktionalitäten, wie die Validierung der Beschreibungsdateien, nicht mehr ermöglichten. Diese Funktionalitäten konnten für den Prototypen jedoch ausser acht gelassen werden, da unsere Beschreibungsdateien auch der von uns festgelegten Syntax

entsprachen. In einer weiteren Entwicklungsstufe sollten solche Funktionalitäten jedoch berücksichtigt werden.

Es wurde hierfür auch ein *CVS*[1] Zugang eingerichtet, um interessierten Entwicklern den Zugang zu den Quelldaten zu ermöglichen, damit eine Mitarbeit an diesem Projekt und eine Weiterentwicklung erfolgen kann. Weitere Informationen hierzu findet man direkt auf dem Projektserver `http://automatisierung.fh-pforzheim.de` im *offenen Labor*.

[1] Concurrent Versioning System

5 Epilog

5.1 Zusammenfassung

In dieser Projektarbeit wurde keine Software, sondern mehr eine Methode zur Beschreibung von Netzwerkstrukturen in der Automatisierungstechnik geschaffen, welche die Visualisierung der Daten aus dem Prozess ermöglicht. Automationszellen, welche mit Hilfe dieser Methode beschrieben wurden, lassen sich sofort im Internet visualisieren und deren Daten weiter verarbeiten. Die Entwicklung der grundlegenden Anforderungen an ein Datenerfassungssystem, wie die Realisierung einer Datenbankanbindung, etc., wird vom Automatisierungsportal übernommen. Lediglich die logische Struktur der Automationszelle, sowie etwaige Visualisierungswünsche müssen spezifiziert werden.

5.2 Ausblicke

In den einzelnen Kapiteln sind Ideen der Weiterentwicklung schon teilweise geschildert worden. Die Menge der Einsatzmöglichkeiten und damit auch der Erweiterungen, die auf Basis der vorliegenden Projektarbeit möglich ist, scheint auf den ersten Blick schier unendlich. Allerdings unterliegen diese auch den Einschränkungen, die die Unabhängigkeit der Beschreibung von der eingesetzten Hardware fordert. Unüberlegte Modifikationen führen nur allzu schnell zu einem proprietären System und schränken die Einsatzmöglichkeiten weiter ein.

Weitere Ausbaustufen sind aber in Planung oder befinden sich schon in der Entwicklung. Dazu gehören unter anderem die Entwicklung einer frei beschreibbaren Datenbankanbindung, um Daten auch in vorhandene Datenbanksysteme ablegen zu können, sowie die Entwicklung eines validierenden Parsers für die Beschreibungsdateien, um Fehler frühzeitig erkennen und beheben zu können.

Um die Praxistauglichkeit näher untersuchen zu können, wird im Rahmen einer weiteren Projektarbeit eine Wetterstation auf Basis dieses Systems entworfen.

Intensive Weiterentwicklung des Systems kann diesem schon bald die Tore zur industriellen Nutzung öffen. Ein Blick ins Internet zeigt, dass Systeme dieser Art sehr grosses Interesse auf sich ziehen. Es werden an vielen Stellen Systeme zur Beschreibung von Prozessdaten entwickelt, doch der Gedanke, die logischen Zusammenhänge unabhängig von der eingesetzten Systemtechnik frei zu beschreiben und diese dann über das Internet zu visualisieren, wird nur in sehr wenigen Fällen verfolgt. Meist handelt es sich um spezialisierte Systeme.

> Wenn ein gewisses technisches Können erreicht ist, verschmelzen Wissenschaft und Kunst gern zu Ästhetik, Bildhaftigkeit und Form. Die grössten Wissenschaftler sind immer auch Künstler.
>
> Albert Einstein, 1923

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[Mic99] Thomas Michel. *XML Kompakt.* Carl Hanser Verlag, München, 1999.

[MS98] M. Wielsch M. Sylvester, M. Weber. *Linux intern.* Data Becker GmbH & Co. KG, Düsseldorf, 1998.

[Sch00] Dennis Schaaf. *Perl.* bhv Verlags GmbH, Kaarst, 2000.