Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Design und Implementierung einer Bibliothek für Webbrowser-basierte Bedienoberflächen von Mess- und Regelanwendungen auf Ethernet-fähigen Mikroprozessorsystemen

vorgelegt von:
Stefan Diercks <stefan.diercks@informatik.haw-hamburg.de>

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Stefan Diercks

Design und Implementierung einer Bibliothek für Webbrowser-basierte Bedienoberflächen von Mess- und Regelanwendungen auf Ethernet-fähigen Mikroprozessorsystemen

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Hans H. Heitmann
Zweitgutachter : Prof. Dr. rer. nat. Kai von Luck
Abgegeben am 15. August 2008

Stefan Diercks

Thema der Diplomarbeit

Design und Implementierung einer Bibliothek für Webbrowser-basierte Bedienoberflächen von Mess- und Regelanwendungen auf Ethernet-fähigen Mikroprozessorsystemen

Stichworte

Messtechnik, Regelungstechnik, Webbrowser, Ethernet, Mikroprozessor, Java, Applet, Grafische Benutzeroberflächen, Prozessdatenvisualisierung, uIP, ARM7TDMI, LPC2468

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Entwicklung einer Bibliothek zur einfachen Integration einer Webbrowser-basierten Benutzeroberfläche in bestehende Systeme der Mess- und Regelungstechnik. Die Bibliothek basiert auf der engen Kopplung zwischen einem C-Modul und einem Java-Applet, welches über den freien IP-Stack μIP im Netzwerk verfügbar gemacht wird. Die Bibliothek bietet sowohl ein Eventsystem, über das grafische Bedienelemente transparent in bestehende Anwendungen integriert werden können, als auch eine grafische Messdatenvisualisierung, welche die Auswertung der Prozessdaten des Systems ermöglicht. Das nachträgliche Hinzufügen von eigenen Bedienelementen und Darstellungsarten für die Visualisierung ist Teil des Konzepts und wird durch einheitliche Schnittstellen in jeder Hinsicht unterstützt. Der Zugriff auf die Bedienoberfläche ist über ein hierarchisches Zugangssystem abgesichert, welches den Zugriff auf unterschiedliche Bedienkontexte des Systems regelt. Für die Verwendung der Bibliothek ist kein Betriebssystem erforderlich.

Stefan Diercks

Title of the paper

Design and implementation of a library for browser-based user interfaces for measurement - and control applications on ethernet-capable microprocessor systems

Keywords

Control systems, measurement applications, browser, ethernet, microprocessor, Java, Applet, graphical user interfaces, process visualisation, uIP, ARM7TDMI, LPC2468

Abstract

This thesis deals with the development of a library for the easy integration of a web-based user interface into existing systems of measurement and control technology. The library is based on the close coupling between a C-module and a Java Applet, which is served via the free IP-Stack μIP. The library offers both, an event system, to integrate graphic controls seamlessly into existing applications, as well as graphical data visualization, to support the evaluation of the systems process data. It allows to serve different control contexts, based on a hierarchical access scheme.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Der Entwurf von Hardwareplattformen zur Realisierung von autonomen Mess- und Regelanwendungen ist durch die breite Palette an leistungsfähigen Mikroprozessoren ein kostengünstiges Unterfangen. Aktuelle Mikroprozessoren bieten eine Vielzahl an Funktionseinheiten und Schnittstellen mit denen die Bedürfnisse dieser Klasse von Anwendungen sehr weit abgedeckt sind. Viele Anwendungen benötigen nicht viel mehr zusätzliche Komponenten als die zur Erfassung von Messdaten erforderlichen Sensoren, sowie einige elektronische Bauelemente. Sobald allerdings ein Bedarf an Visualisierungskomponenten zur Benutzerinteraktion besteht, deren Informationsgehalt über das Maß an Information hinausgeht, die durch ein paar Dioden oder ein kleines Display dargestellt werden können, wird sehr schnell ein größeres Display sowie eventuell eine leistungsfähigere CPU benötigt. Hierdurch entstehen weitere Randbedingungen wie Mehrkosten und erhöhter Energiebedarf. Nicht zuletzt aus diesen Gründen sind Alternativen gefragt, welche diesen Bedarf erfüllen, ohne die Kosten oder den Energieverbrauch maßgeblich zu beeinträchtigen.
Bei vielen Systemen wird die erweiterte Benutzerinteraktion auf einen externen Rechner ausgelagert, der bei Bedarf über eine Schnittstelle mit der Plattform verbunden werden kann. Hierzu kann man zum Beispiel einen seriellen Port verwenden. Serielle Ports gehören allerdings schon länger nicht mehr zur Grundausstattung moderner Rechner. In der heutigen vernetzten Welt verfügt jeder zeitgemäße Rechner über eine Netzwerkschnittstelle und hat einen Browser installiert. Das Bedürfnis an Vernetzung ist seitdem stetig gestiegen. Es besteht der Wunsch nach allgegenwärtiger Verfügbarkeit von Informationen und der Eingliederung beliebiger Systeme in das Netz. Diese Entwicklung macht auch vor Mikroprozessoren nicht halt. Inzwischen sind diverse Modelle mit integrierter Netzwerkschnittstelle zu bekommen, welche durch den niedrigen Preis auch auf dem Gebiet der autonomen Mess- und Regelanwendungen zu verstärkter Innovation geführt haben.

Die in dieser Arbeit entstandene Bibliothek richtet sich an Entwickler von technischen Anwendungen auf Mikroprozessoren, die einen schnellen Einstieg in die Entwicklung von Benutzeroberflächen wünschen. Viele gebräuchliche Ansätze der konventionellen Web-Technologie versagen besonders auf Plattformen des Niedrigpreissegments, wo Prozessoren mit sehr wenig Ram eingesetzt werden. Im Bereich der Mess- und Regelanwendungen sind Prozessdatenvisualisierungen gefragt, welche einen schnellen Überblick über die Systemparameter ermöglichen. Ebenso besteht ein Bedarf für universelle Bedienkomponenten, die die Anforderungen an eine moderne Benutzeroberfläche erfüllen. Die Programmierung von Webbrowser-basierten Benutzeroberflächen auf Mikro-

prozessorsystemen stellt Entwickler vor besondere Herausforderungen. Die in dieser Arbeit vorgestellte Bibliothek stellt sich der Aufgabe, die notwendigen Mechanismen zur einfachen Realisierung anwendungsbezogener Benutzeroberflächen bereitzustellen und dabei genügend Rechenzeit für die eigentliche Anwendung übrig zu lassen. Im Laufe der Arbeit entsteht ein Konzept, welches die einfache Anbindung und integration einer gestaltbaren browserbasierten Bedienoberfläche in bestehende Anwendungen ermöglicht. Durch den in dieser Arbeit verwendeten Fat-Client Ansatz steht der Anwendung die Rechenpower des Clients zur Seite, wenn es um die Aufbereitung der Daten zur Visualisierung geht.

2 Aufgabenstellung

2.1 Grundlegendes

Das Anwendungsspektrum von Systemen der Mess- und Regelungstechnik ist sehr groß. Genauso unterschiedlich wie die Anwendungen sind auch deren Größe und Ausprägung. Die Anzahl an Mess- und Regelgrößen variiert von Anwendung zu Anwendung. In der Fertigungstechnik trifft man oft auf verteilte Anwendungen, hier werden einzelne Produktionsprozesse über einen *Feldbus*[5] oder ein Netzwerk zusammengeschlossen und an zentraler Stelle überwacht und gesteuert. Diese Arbeit beschäftigt sich mit der Webbasierten grafischen Darstellung von Oberflächen für autonome Mess- und Regelanwendungen auf Mikrocontroller-basierten Systemen. Im Gegensatz zu ausgewachsenen Produktionsanlagen befinden hier sich sowohl der Kern der eigentlichen Anwendung, als auch zumindest Teile der Benutzerinteraktion und Prozessdatenvisualisierng auf einem einzigen Mikrocontroller. Einzig die grafische Bedienoberfläche dieser Anwendung wird über die Netzwerkschnittstelle des Mikrocontrollers verfügbar gemacht, um auf einem weiteren Rechner dargestellt zu werden.

2.2 Ziel der Arbeit

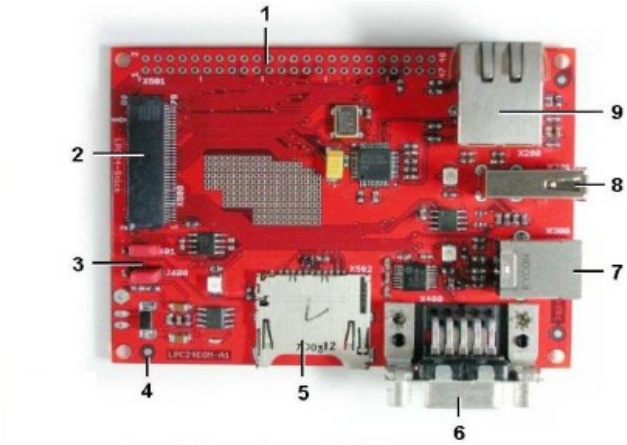
Diese Arbeit befasst sich mit der Entwicklung einer Bibliothek für Webbrowserbasierte grafische Bedienoberflächen von Mess- und Regelanwendungen auf Ethernet-fähigen Mikroprozessorplattformen. Das Ziel dieser Arbeit ist die Bereitstellung von Mechanismen zur Benutzerinteraktion und der Datenvisualisierung der Prozessdaten einer Mess- und Regelanwendung in einem Browser. Ein weiteres Ziel dieser Arbeit ist, dabei möglichst plattform- und betriebssystemunabhängig zu funktionieren, um auch nachträglich möglichst einfach in bereits bestehende Systeme integriert werden zu können. Es kann das gesamte Spektrum aktueller Webtechnologien verwendet werden. Hierbei sind natürlich Technologien die eine hohe Last auf der CPU erzeugen, wie zum Beispiel die serverseitige Generierung von Grafiken, ausgeschlossen. Bei der Entwicklung ist auf geringstmöglichen Ressourcenbedarf zu achten, um möglichst kleine Mikroprozessoren verwenden zu können. Durch den Einsatz von Ethernet ist natürlich eine räumliche Distanz zwischen der Plattform und einem Client-Rechner möglich, im Vordergrund steht jedoch die Verfügbarkeit von Rechenzeit und einem Display auf der Client-Seite. Sobald ein System

in ein Netzwerk oder etwa das Internet integriert wird, steigen die Anforderungen auf dem Gebiet der Netzwerksicherheit beträchtlich. Diese speziellen Anforderungen spielen in dieser Arbeit eine eher untergeordnete Rolle. Ein typischer Anwendungsfall ist eher die direkte Verbindung des Systems mit einem Client-Rechner.

2.3 Plattform

Als Plattform kam hierbei ein Entwicklerkit der Firma *Hitex Development Tools* zum Einsatz. Die Plattform basiert auf dem *LPC2468*[33] von *NXP*[1], einem Mikroprozessor auf Basis der *ARM7TDMI-S*[22] - Architektur. Der Prozessor verfügt über einen integrierten Netzwerkcontroller. Der Prozessor kann mit einem Takt von bis zu 72 MHz betrieben werden und verfügt über 512 Kilobyte Flash-Speicher und 16 Kilobyte Ram.

Abbildung 2.1: Der LPC-Stick ©Hitex Development Tools



1	IO-Connector pads (X501)
2	Stick-IO-Connector (X500)
3	Pads for - external power connection +5V/GND (JP500) - external power enable (JP601) - external power enable via USB-B/VBUS (JP600)
4	External power connector (JP501)
5	MiniSD card (X502)
6	RS232 Connector (X200)
7	USB-A connector (X300)
8	USB-B connector (X3001)
9	LAN Connector (X200)

Abbildung 2.2: Das LPC-Stick Com Board ©Hitex Development Tools

3 Grundlagen

3.1 Vorhandene Lösungsansätze

3.1.1 Softwarelösungen

Bei Mess- und Regelanwendungen und Prozessvisualisierung in Zusammenhang mit Computern steht schnell der Name *Labview*[6] im Raum. Labview ist eine weit verbreitete grafische Programmiersprache von National Instruments zur Steuerung und Visualisierung. Seine Vorteile liegen in der einfachen Programmierung über graphische Elemente und die vielfältige Unterstützung durch Hersteller von Messhardware. Bei der Verwendung von Labview werden die kompletten Mess- und Regelschleifen auf dem PC abgebildet und entsprechend visualisiert. Im Regelfall geschieht dies über spezielle Hardware, die direkt an den PC angeschlossen wird, die Einbindung eines Mikrocontrollersystems ist hier nicht direkt vorgesehen. Mit speziellen Erweiterungen kann Labview-Code auch für embedded-Systeme compiliert werden, dies beschränkt sich allerdings auf Regelkreise und ähnliches, eine Anbindung dieses Codes an Labview ist nicht direkt vorgesehen. Der schnelle Einsatz eines Clientrechners im Sinne eines grafischen Terminals auf Labview-Basis ist zunächst mit der Installation und Konfiguration der Labview-Umgebung auf einem Client-Rechner verbunden, wobei für jeden Arbeitsplatz eine konstenpflichtige Lizenz benötigt wird. Die Anforderungen an diese Arbeit kann Labview daher nicht erfüllen. Ähnliches gilt für verwandte Systeme wie z.B. Agilent *Vee*[4] oder Testpoint von Keithley.

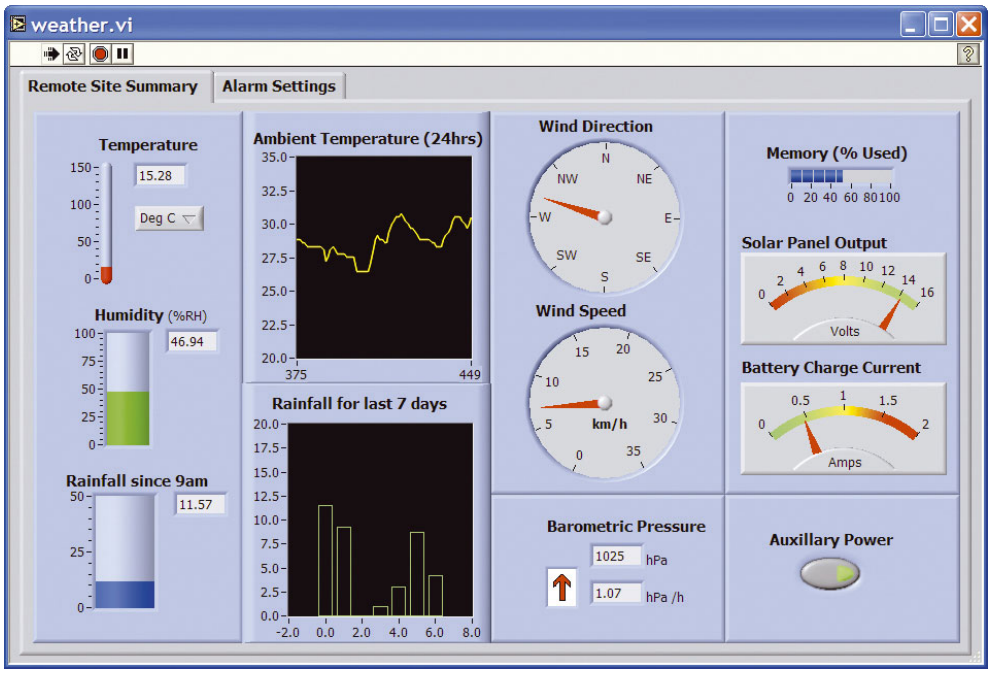


Abbildung 3.1: Beispiel für eine Labview Visualisierung Quelle: datataker.com

Von National Instruments wird auch das Programm *Diadem*[2] vertrieben. Dabei handelt es sich um ein kostenpflichtiges Visualisierungstool für Messwerte, mit dem auch umfangreiche mathematische und statistische Analysen möglich sind. Für eine Datenaufnahme ist Diadem nicht geeignet, hierfür wird auf Labview verwiesen. Diadem kann Log-Dateien, die etwa als Komma-separierte Datenreihen vorliegen, auswerten und visualisieren. Eine Steuerung der laufenden Mess- und Regelanwendung ist nicht möglich. Die Anforderungen an diese Arbeit kann auch Diadem daher nicht erfüllen.

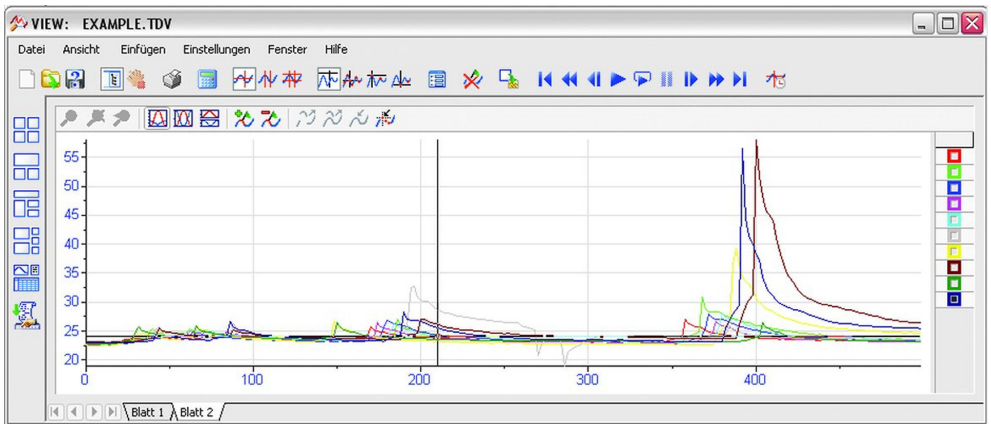


Abbildung 3.2: Beispiel für eine Diadem Visualisierung Quelle: National Instruments

Die vorgestellten Programme sind ausgewachsene Analysewerkzeuge, die mit ihrer Vielzahl an Auswertungsmöglichkeiten den Bereich der Forschung, Fertigungsmesstechnik, Produktevaluierung und Programmierung von Anwendungen diverser Art ermöglichen. Die Installation und Konfiguration auf einem

Client-Rechner und ein tiefergehendes Verständnis dieser Produkte ist unumgänglich.

3.1.2 Hardwarelösungen

Moderne Mess- und Regelanwendungen verwalten große Datenmengen in digitaler Form. Normalerweise geschieht dies intern im Prozessor, es werden nur die wichtigsten Werte über Displays oder LEDs zur Anzeige gebracht. Eine Darstellung auf grafischen Displays mit Kurvenformen, verschiedenen Schriftarten und grafischen Bedienelementen wird oft auf Embedded-Linux-Plattformen realisiert. Die Darstellung auf dem Display wird beispielsweise durch QT™[3] ermöglicht. Anwendungen, die nur zur Darstellung einer visuellen Benutzeroberfläche auf solch eine Plattform wechseln, müssen verhältnismäßig hohe Stückkosten in Kauf nehmen. Hier entstehen große Anforderungen an die Systemressourcen. Es werden größere Prozessoren notwendig, da die benötigte Rechenzeit für die Darstellung auf dem Display durch das System aufgebracht werden muss. Zusätzlich wird ein Display benötigt, welches außerdem die Abmessungen der Plattform maßgeblich beeinflussen kann.

3.1.3 Mögliche Einsatzgebiete

Durch das Vordringen von Netzwerken in den privaten Bereich (zum Beispiel auch durch Home-AV) aber auch durch den vermehrten Einsatz von Ethernet im betrieblichen Umfeld eröffnen sich neue Kommunikationswege. Der Einsatz der Netzwerkschnittstelle zur Übermittlung von Mess- und Regeldaten ist in vielen Bereichen bereits gängige Praxis. Oft wird hierbei mit proprietären Systemen und spezieller Windows-Software gearbeitet um die Daten zu visualisieren, da die Einbringung eines Webservers in eine Plattform grundsätzlich die Komplexität der Web-Entwicklung in Bereiche einstreut, welche sich bisher nur mit reinen Ingenieursdisziplinen beschäftigt haben. Zur Visualisierung von Messdaten müssen ein spezielles Protokoll und weitere Software entwickelt werden, um eine grafische Darstellung auf einem externen Rechner zu ermöglichen, welcher diese Software zunächst installieren muß.

Der Einsatz eines Browsers ermöglicht den einfachen Zugriff auf Netzwerkgebundene Systeme, wobei Entfernungen hierbei durch IP-Netzwerke stark relativiert werden. Die Installation von proprietären Windows-Anwendungen ist ein Ärgernis, wenn man schnell mal aus dem Urlaubsort auf die Daten der heimischen Solaranlage zugreifen möchte.

Solaranlage

Ein Anwendungsbeispiel ist die Steuerungseinheit einer Solaranlage. Bei Solar- und auch Heizungsanlagen wird oft eine separate Steuerung im Wohnraum angebracht. Die Firma Buderus bietet für diesen Anwendungsfall zwei verschiedene Modelle an. Die Buderus Logamatic RC20 (Abb. 3.3)ist ein kostengünstiges System, welches über eine einzeilige Textanzeige und wenige Steuertasten verfügt.

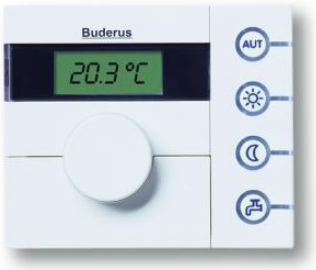


Abbildung 3.3: Buderus Logamatic RC20 (Buderus)

Deutlich aufwendiger ist die Buderus Logamatic RC35 (Abb. 3.4)gestaltet. Dieses Modell verfügt über eine grafische Anzeige, mehrere Tasten (die unter einer Abdeckung verborgen sind) und ein Jog-Dial. Bei dieser Steueranlage ist die Fernkonfiguration und Überwachung der Anlage möglich.

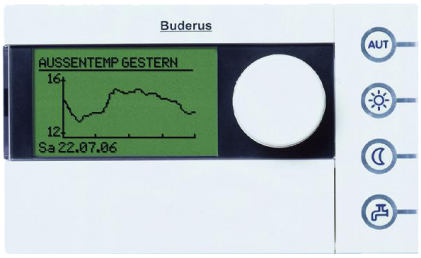


Abbildung 3.4: Buderus Logamatic RC35 (Buderus)

Eine Konfiguration und Überwachung der Anlage über das graphische Display erfordert meist ein gründliches Studium des Handbuchs und bietet nur eingeschränkte Möglichkeiten zur Interaktion. Eine Kombination aus einem kostengünstigen Anzeigesystem im Wohnraum und einer Web-basierten Applikation im Hauptgerät kann einen deutlich höheren Bedienkomfort bieten. Dabei besteht die Möglichkeit, Diagramme und Schaltflächen an einem gemütlichen Arbeitsplatz auf einem angemessen großen Bildschirm mit interaktiven Dialogmöglichkeiten zu bieten und dem Benutzer so die Bedienung zu erleichtern. Der Zugriff von Service- und Wartungspersonal über eine höhere Authentifizierungsstufe würde die Servicefreundlichkeit erhöhen. Eine Anbindung über PNA oder vorhandenen Netzwerk-Infrastrukturen ist vorstellbar, so daß der Hauseigentümer das System als Teil seiner Web-Experience nutzen kann.

3.1.4 Abwägungen

Es gibt selbstverständlich bereits autonome Systeme, die eine Bedienoberfläche im Netzwerk verfügbar machen. Auch die Verwendung eines Browsers haben viele dieser Systeme gemein. Hierbei geht allerdings jeder seinen eigenen Weg. Die Entwicklung von Konzepten zur Darstellung von Anwendungsbezogenen Informationen im Browser bedeutet, einen Webserver zu integrieren und ein Konzept für die Realisierung der Oberfläche zu entwickeln. Dies verlängert die Entwicklungszyklen, da die Verwendung von Webtechnologien für viele Unternehmen immer noch Neuland ist. Der Umfang hat sich bei den meisten Produkten auf die Anforderungen der spezifischen Anwendung beschränkt.

Die zuvor beschriebenen Softwarelösungen sind nicht für das eben beschriebenen Einsatzgebiet gedacht und bringen die Notwendigkeit einer Installation, das Verständnis umfangreicher komplexer Softwarewerkzeuge, und generell zusätzliche Kosten mit sich. Eine Benutzeroberfläche für Mess- und Regelanwendungen stellt sicher nicht die gleichen Anforderungen an ein Softwarewerkzeug, welches im Bererich der Forschung, Entwicklung oder Fertigungsmesstechnik anzusiedeln ist. Es gibt aber eine Schnittmenge. Diese Arbeit versucht, gezielt diese Schnittmenge abzudecken und die kostengünstige und einfache Realisierung von Benutzeroberflächen autonomer Mess- und Regelanwendungen im Bereich kleinstmöglicher Mikroprozessoren zu ermöglichen.

3.2 Mess- und Regelanwendungen

Um den Sinn und Zweck von Mess- und Regelungsanwendungen zu verstehen, ist es zunächst notwendig, die wichtigsten Begriffe aus diesem Anwendungsfeld kurz zu erklären.

3.2.1 Mess- und Regelanwendungen allgemein

Die *Messtechnik* [7] befasst sich mit Geräten und Methoden zur Bestimmung (Messung) physikalischer Größen wie beispielsweise Länge, Gewicht, Kraft, Druck, elektrischer Strom, Temperatur oder Zeit. Wichtige Teilgebiete der Messtechnik sind die Entwicklung von Messsystemen und Messmethoden, sowie die Erfassung, Modellierung und Reduktion (Korrektur) von Messabweichungen und unerwünschten Einflüssen. Dazu gehört auch die Justierung und Kalibrierung von Messgeräten. Die Messtechnik ist in Verbindung mit Steuerungs- und Regelungstechnik eine Voraussetzung der Automatisierungstechnik. Für die Methoden und Produkte der industriellen Fertigung kennt man den Begriff der Fertigungsmesstechnik.

Die *Regelungstechnik*[10] ist ein Teilgebiet der Automatisierungstechnik. Sie

befasst sich mit der gezielten Beeinflussung von physikalischen, chemischen, biologischen oder anderen Größen in verschiedensten Geräten und Anlagen. Häufig ist die vollständige Automatisierung eines solchen Vorgangs möglich, jedoch muss gelegentlich auch menschliche Tätigkeit aus technischen und/oder wirtschaftlichen Gründen zum Vorgang beitragen (Elementares Beispiel: Einstellen einer angenehmen Duschwassertemperatur, komplexeres Beispiel: Lenken eines Passagierflugzeugs).

Mess- und Regelanwendungen befassen sich grundsätzlich mit der Erfassung von Sensordaten, um Regelgrößen in Abhängigkeit dieser Sensordaten mittels eines Regelungsalgorythmus gezielt zu beeinflussen. Hierbei dienen häufig *Regelkreise*[9] als Beschreibungsmodelle für die Regelung. Ein Regelkreis ist ein rückgekoppeltes System, welches mindestens aus einer Regelungsstrecke, einem Regler und der Rückführung besteht. Kennzeichnend für einen Regelkreis ist der geschlossene Wirkungskreis mit einer negativen Rückkopplung. Regelkreise werden mathematisch mit Hilfe der Systemtheorie beschrieben, die parallel zur Regelungstechnik entwickelt wurde. Diese Theorie vermag sowohl zeitkontinuierliche als auch zeitdiskrete Systeme und Signale zu beschreiben.

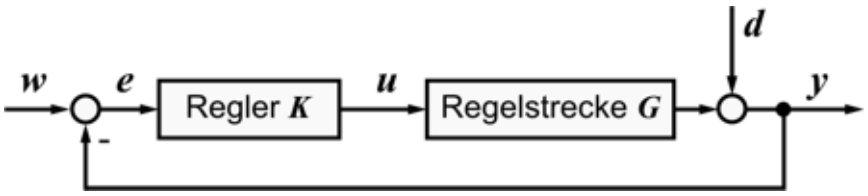


Abbildung 3.5: Ein Standardregelkreis Quelle: Wikipedia

Regelkreise werden in der Technik verwendet, wenn das Verhalten der Regelstrecke nicht den Anforderungen genügt. Dazu wird der Regler so entworfen, dass der Regelkreis das gewünschte Verhalten möglichst gut annimmt. Das gewünschte Verhalten kann vielfältig sein. Beispielsweise kann das Ziel in der Stabilisierung einer instabilen Regelstrecke bestehen. Eine weitere übliche Forderung, die Sollwertfolge, verlangt, dass der Ausgang **y** dem Sollwert **w** asymptotisch folgen soll. Abbildung 3.5 zeigt das Blockschaltbild einen einfachen Standardregelkreises bestehend aus der Regelstrecke 'G', dem Regler 'K' und einer negativen Rückkopplung der Regelgröße 'y' (auch: Istwert) auf den Regler. Die Regeldifferenz 'e' wird aus der Differenz zwischen der Führungsgröße 'w' (auch: Sollwert) und der Regelgröße errechnet. Der vom Regler ermittelte Stellwert 'u' wirkt auf die Strecke und damit wiederum auf die Regelgröße ein. Die Störgröße d bewirkt eine Veränderung der Regelgröße, die nicht gewünscht ist und kompensiert werden muss.

Regler[11] beeinflussen selbsttätig in einem meist technischen Prozess eine oder mehrere physikalische Größen auf ein vorgegebenes Niveau unter Reduzierung von Störeinflüssen. Ihre Behandlung ist Kern der *Regelungstechnik*. *Regler* vergleichen innerhalb eines Regelkreises laufend das Signal des Sollwertes mit dem gemessenen und zurückgeführten Istwert der Regelgröße und ermitteln aus dem Unterschied der beiden Größen (*Regeldifferenz*) eine Stellgröße, wel-

che die Regel-Strecke so beeinflusst, dass die Regelabweichung spätestens im eingeschwungenem Zustand zu einem Minimum wird.

Der Einsatz von Reglern ist so vielfältig und unterschiedlich, wie es Regelaufgaben aus allen Bereichen des Haushaltes, der Industrie, der Luft- und Raumfahrt, Forschung usw. gibt. Dieser Abschnitt soll nur einen kleinen Einblick in die Grundlagen der Mess- und Regelanwendungen geben und erhebt keinen Anspruch auf Vollständigkeit.

3.2.2 Mess- und Regelanwendungen auf Mikroprozessoren

Begrenzter Speicher

Mikroprozessorsyteme verfügen meist über eine überschaubare Menge an Arbeitsspeicher. Der in dieser Arbeit verwendete *LPC2468*[33] verfügt beispielsweise über 64 Kilobyte Arbeitsspeicher. Obwohl dies im Vergleich zu ausgewachsenen Desktoprechnern eine verschwindend geringe Menge an Speicher ist, ist dies eine eher typische, eventuell sogar überdurchschnittliche Menge an Arbeitsspeicher für einen Mikroprozessor. Trotzdem ist Arbeitsspeicher auf einer solchen Plattform grundsätzlich ein knappes Gut, welches wohlüberlegt eingesetzt werden sollte. Im Gegensatz hierzu fasst der Programmspeicher des Prozessors (Von- Neumann- Architektur) satte 512 Kilobyte Flash-Speicher, welcher ohne Umwege im gleichen Adressraum ansprechbar ist.

Scheduling-Verfahren und Interrupts

Mess- und Regelanwendungen auf Mikroprozessorsystemen tasten regelmäßig Eingänge ab und berechnen aufgrund eines Regelungsverfahrens Regelgrößen. Diese Regelgrößen werden direkt oder indirekt verwendet, um Ausgänge zu regeln. Alle Systeme führen Anwendungscode aus, welcher in verschiedene Arbeitsschritte unterteilt wurde, um diese in einer zeitlichen Abfolge koordiniert auszuführen. Die einfachste Vorgehensweise ist sicherlich die Superloop-Architektur[28, S.164], bei der alle nötigen Arbeitsschritte innerhalb einer globalen Endlosschleife ausgeführt werden. Viele Anwendungen benötigen jedoch spezielle Scheduling-Mechanismen um etwa die Häufigkeit der Ausführung einzelner Subroutinen dynamisch zur Laufzeit zu variieren oder etwa zeitliche Garantien geben zu können. Ereignisgesteuerte Systeme machen Gebrauch von Interrupts, in denen zeitnah auf externe Ereignisse wie etwa das Auslösen eines Endabschalters reagiert werden muss. Es gibt Regelanwendungen, die aufgrund eines sehr langsamen Prozesses, wie beispielsweise der Erhitzung eines Boilers, keine harten Anforderungen an das Zeitverhalten der Regelanwendung stellen. Es gibt allerdings auch Regelanwendungen wie zum Beispiel eine einfache Geschwindigkeitsregelung mittels einer Drehzahlmessung über einen Hal-Sensor. Damit die Geschwindigkeit geregelt werden kann muss die

Anzahl der Umdrehungen, die in einem gewissen Zeitraum stattfinden, ermittelt werden. Dies ist zwar auch noch keine High-End-Anwendung, aber es zeigt, das bei manchen Systemen die genaue Spezifikation und Einhaltung eines Zeitverhaltens notwendig ist. Solche Anwendungen verwenden meist spezielle Scheduling-Verfahren, um die konkreten Anforderungen an die Regelanwendung zu erfüllen. Da eine Bibliothek nur eine unterstützende Funktion bei der Ausführung der eigentlichen Aufgabe des jeweiligen Systems hat, sollte sie möglichst wenige Anforderungen an das Scheduling-Verfahren des zugrundeliegenden Systems stellen.

3.3 IP-Stacks

3.3.1 IP-Stacks allgemein

Durch den Erfolg des Internet ist IP[1] zu einem globalen Standard für Kommunikation geworden. Ein Mikroprozessorsystem, dass mit einem Netzwerkcontroller ausgestattet ist, kann an ein lokales Netzwerk oder sogar an das Internet angeschlossen werden. Durch die Unterstützung von TCP/IP kann es direkt mit anderen Teilnehmern im Netzwerk kommunizieren. Um die verschiedenen Bestandteile eines IP-Stacks zu beschreiben, werden häufig das *OSI-Modell*[8] und das *TCP-IP Referenzmodell*[12] verwendet. Abbildung 3.6 zeigt beide Modelle und die dazugehörigen Anwendungs-, Vermittlungs- und Transportschichten. Die abgebildeten Schichten machen jeweils Gebrauch von den darunterliegenden Schichten. Die unterste Schicht ist die physikalische Schicht, die oberste Schicht ist die Anwendungsschicht.

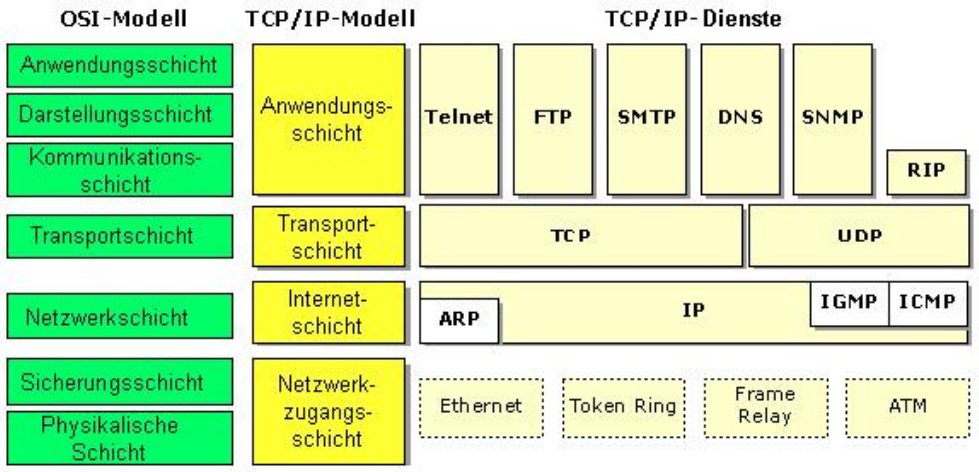


Abbildung 3.6: Beschreibungsmodelle für Netzwerkkommunikation

Die grundsätzliche Aufgabe eines IP-Stacks ist die Bereitstellung von Mechanismen, die die Netzwerkkommunikation in einem IP-Netzwerk ermöglichen. Zur Kommunikation mit dem IP-Stack stellt dieser der Anwendung sogenannte *Sockets* zur Verfügung. Eine Anwendung kann mittels der Anwendungsschnittstelle des Sockets über das Netzwerk kommunizieren. Ausgewachsene IP-Stacks bieten meist sogenannte *Berkeley Sockets*. Die Berkeley Socket API(auch bekannt als BSD-Socket-API) ist eine Bibliothek zur Entwicklung von Netzwerkanwendungen in der Programmiersprache C. Sie ist ursprünglich durch die Computer Systems Research Group der Berkeley-Universität in Californien als Bestandteil eines Unix-Derivats, der Berkeley Software-Distribution, veröffentlicht worden. Heute gilt die abstrakte Form der Berkeley-Socket-API als de-facto Standard für Socket-Anwendungsschnittstellen. Viele Socket-Anwendungsschnittstellen sind in Anlehnung an diese Schnittstel-

[1] *Internet Protocol*

le entwickelt worden. Anwendungen machen durch den Socket Gebrauch von der darunterliegenden Transportschicht, die wiederrum auf das IP-Protokoll aufsetz. Auf der Transportschicht findet meist TCP[2][13] oder UDP[3][29] Verwendung. Im Folgenden werden die beiden wichtigsten Vertreter der Transportprotokolle kurz erklärt.

UDP

UDP ist ein minimales, verbindungsloses Transportprotokoll, welches die Zuordnung von Datenströmen im Netzwerk zu einer konkreten Anwendung ermöglicht. Hierzu werden Portnummern verwendet. Verbindungslose Transportprotokolle geben keine Garantie dafür, dass ein gesendetes Paket auch ankommt oder dass die Pakete in der Reihenfolge ankommen, in der sie gesendet wurden. Ein Sendevorgang ist für den Stack abgeschlossen, sobald die Daten die Netzwerkschnittstelle verlassen haben. UDP wird unter anderem zur Übertragung von Broadcasts verwendet. Broadcasts haben die Eigenschaft, dass sie von mehreren Rechnern gleichzeitig empfangen werden können. Dies funktioniert nur aufgrund der Tatsache, dass keine spezielle Antwort eines Empfängers notwendig ist(Verbindungslosigkeit). Anwendungen, die auf UDP aufsetzten, müssen selbst Mechanismen zur Datenflusskontrolle oder Neuübertragung auf Anwendungsebene vorsehen, um verlässliche Datenströme auf UDP-Basis zu realisieren.

TCP

TCP hingegen stellt zuverlässige Vollduplexdatenströme auf dem IP Layer bereit. Da die IP-Schicht zwischen dem Sender und dem Empfänger Pakete verwerfen oder sie in unterschiedlicher Reihenfolge ausliefern kann, verwendet TCP Sequenznummern und automatische Neuübertragungen um der Anwendung verlässliche Datenübertragungen bereitzustellen. Dieses Transportprotokoll funktioniert als Punkt-zu-Punkt Verbindungen zwischen zwei Systemen. Im TCP Protokoll sind verschiedene Mechanismen zur Datenflusskontrolle, wie zum Beispiel die Bestätigung von eingegangenen Paketen spezifiziert. Diese Komplexität verbirgt sich allerdings vor dem Anwender. Es ist die Aufgabe des TCP/IP-Stacks, verlorengegangene Pakete neu zu übertragen, falls versendete Pakete nicht innerhalb einer gewissen Zeit bestätigt wurden. Die Gegenseite stellt durch Lücken in den Sequenznummern der eintreffenden Pakete fest, dass Pakete fehlen und wartet mit der Auslieferung der Daten, bis diese lückenlos und in der richtigen Reihenfolge vorliegen. TCP/IP-Stacks müssen zu diesem Zweck alle gesendeten Pakete aufbewahren, bis sie bestätigt wurden. Die Speicherung dieser Paketdaten ist als einer der kritischen Faktoren bei der Verwendung von TCP auf Mikroprozessorsystemen einzustufen.

[2] *Transmission Control Protocol*
[3] *User Datagram Protocol*

3.3.2 IP-Stacks auf Mikroprozessorsystemen

Traditionelle TCP/IP Implementierungen sind verhältnismäßig ressourcenhungig. Die hohen Systemanforderungen haben es lange Zeit unmöglich gemacht, einen vollständigen TCP/IP Stack auf Systemen mit beispielsweise 16 Kilobyte Ram und 64 Kilobyte Rom zu realisieren. Inzwischen gibt es IP-Stacks für mikroprozessorbasierte Syteme, welche die Anbindung eines externen oder integrierten Netzwerkchips ermöglichen. Um die Codegröße und den Overhead zu minimieren, der bei der Implementierung des vollen Umfangs eines IP-Stacks benötigt wird, werden hierzu häufig einzelne Mechanismen, die für den Betrieb von TCP/IP nicht unbedingt notwendig sind, entfernt. Einige Produkte verzichten beispielsweise auf gewisse Mechanismen in der Anwendungsschnittstelle des IP-Stacks, wenn diese nicht unbedingt für die Abwicklung der Netzwerkkommunikation sind. Andere Produkte verzichten auf die Implementierung von UDP oder einiger ICMP-Mechanismen. Einige dieser Einschränkungen haben einen nennenswerten Einfluss auf den Datendurchsatz der Netzwerkkommunikation. Hierzu gehören das *Sliding Window Protocoll* und die Technik des *Delayed Acknowledgement*. Diese beiden Techniken werden im Folgenden kurz erklärt.

Sliding Window

Die meisten „ausgewachsenen“ TCP Implementierungen nutzen das *Sliding Window Protokoll* um die Übertragung von Netzwerkpaketen zu beschleunigen. Das Protokoll verwaltet die Daten von gesendeten Paketen bis diese bestätigt wurden, um die Übertragung von weiteren Paketen zu ermöglichen, obwohl zuvor versendete Pakete noch nicht bestätigt wurden. Netzwerkpakete können maximal eine Länge von 1460 Bytes haben. Die maximale Größe für Netzerkpakete zwischen zwei Plattformen wird beim Verbindungsaufbau ermittelt. Wie im vorangegangenen Abschnitt über TCP erklärt wurde, muß der IP-Stack des Senders alle Datenpakete aufbewahren bis diese vom Partner bestätigt wurden. IP-Stacks auf Empfängerseite müssen aber auch Empfangspuffer bereitstellen, um eingehende Pakete zwischenzuspeichern. Hierzu wird durch den Empfänger im sogenannten *Receive Window* bekanntgemacht, wie viele Bytes er gleichzeitig verarbeiten kann. Die Größe des Receive Windows ist die maximale Anzahl an Bytes die an einem Stück (auch in mehreren Paketen) übertragen werden kann. Microsoft Windows verwendet initial beispielsweise standardmäßig ein Receive Window von 8760 Bytes pro Verbindung, was einer Anzahl von 6 vollen Ethernetpaketen a 1460 entspricht. Der Sender stellt die Größe seines Sendefensters auf die Größe des Empfangsfensters ein. Hat das Empfangsfenster beispielsweise die Größe m, kann der Sender m Bytes versenden ohne dabei auf ein Acknowledge zu warten. Danach ist die Empfangskapazität des Empfängers ausgelastet und der Sender muß warten, bis er ein Acknowledge empfängt. Nachdem er das Acknowledge empfangen hat ist wieder Platz im Empfangspuffer des Empfängers und der Sender setzt das Senden fort. Für den Fall dass der Empfänger ein gesendetes Paket nicht bestätigt,

muss der Sender die Daten aller gesendeten Pakete vorhalten. Die Summe aller gleichzeitig auf dem Weg zum Client befindlichen Daten entspricht maximal der Größe des Empfangsfensters des Empfängers. Das Ausschöpfen der vollen Kapazität des Empfangsfensters geht mit der Anpassung des Sendefensters an dessen Kapazität und damit der Bereitstellung eines Speichers für die eventuell neu zu übertragenden Pakete einher.

Der Sender muß die Kapazität des Empfangsfensters des Empfängers allerdings nicht voll ausschöpfen. Ein Sendepuffer von 8 Kilobyte ist für die Verhältnisse eines Mikroprozessors sehr groß. Auf Mikroprozessoren ist Speicher ein knappes Gut. Um selbst auf kleinsten Rechnerarchitekturen funktionieren zu können, verzichten einige Stacks für Mikroprozessorsysteme auf die Verwendung eines Sliding Windows und sorgen dafür, dass immer nur ein Paket zur Zeit auf dem Weg ist. Hierdurch wird nicht nur Speicher gespart, die Anwendungsschnittstelle des Stacks kann stark vereinfacht werden, wenn immer nur jeweils ein Paket unterwegs ist. In diesem Fall kann der Stack das Vorhalten des neu zu übertragenden Pakets sogar der Anwendung überlassen, was weiteren Speicher spart. Bei diesem Ansatz muß jedoch nach jedem Paket zunächst ungefähr die Round-Trip-Time (RTT) gewartet werden, bis das Acknowledge eintrifft. Der Verzicht auf große Puffer geht mit einem verminderten Datendurchsatz einher. Durch die Formel 6.1 kann der zu erwartende Datendurchsatz unter Berücksichtigung der Round-Trip-Time abgeschätzt werden.

$$p = \frac{s}{t} \tag{3.1}$$

p Maximaler Datendurchsatz
s Segmentgröße in Bytes
t Round-Trip-Time RTT

Delayed Acknowledge

Ein TCP Empfänger der *Delayed Acknowledge* verwendet, wartet eine gewisse Zeit, ob die Anwendung eine Antwort an den Sender verschickt, um die Bestätigung in diesem Paket unterzubringen. Wenn kein weiteres Paket innerhalb dieser Zeit eintrifft, sendet er die Bestätigung los. Das Zeitfenster kann teilweise bis zu 500 ms groß sein, eine typische Zeit ist 200 ms. Ein TCP Sender, der nur ein einziges Paket zur Zeit versendet, wird durch dieses Verhalten stark ausgebremst. Da der Empfänger nach jedem Empfang weitere 200-500ms wartet bis er das Paket bestätigt, wird insgesamt deutlich mehr Zeit mit Warten verbracht, als eigentlich für die Datenübertragung benötigt würde. Formel 3.2 berechnet wie Formel 6.1 den maximalen zu erwartenden Datendurchsatz, allerdings erweitert um das im Delayed Acknowledgement verwendete Timeout.

$$p = \frac{s}{t + t_d} \tag{3.2}$$

p Maximaler Datendurchsatz
s Segmentgröße in Bytes
t Round-Trip-Time RTT
t_d Delayed Acknowledge Timeout

Rechenbeispiel:
Segmentgröße: 1460 Bytes (maximale Nutzdaten eines Ethernet-Frames)
Round-Trip-Time: 2 ms (direkte Verbindung)
Delayed Acknowledgement Timeout: 200 ms

$p = \frac{s}{t+t_d} = \frac{1460Bytes}{2ms+200ms} = 7,2KB/s \triangleq$ 57,8 KBit/s

Das gleiche Rechenbeispiel ohne das Delayed Acknowledgement:

$p = \frac{s}{t+t_d} = \frac{1460Bytes}{2ms} = 730KB/s \triangleq$ 5,8 MBit/s

Es ist erwähnenswert, dass Mikroprozessoren, die IP-Stacks verwenden, nicht unbedingt große Datenmengen übertragen müssen. Sobald die Anwendung Pakete in einer langsamen Periode verschickt, ist der Einfluß des Delayed Acknowledgement auf den Datendurchsatz sehr gering bis überhaupt nicht gegeben. Der maximale Datendurchsatz beim Empfang von Daten wird durch diesem Algorithmus überhaupt nicht beeinflusst.

Hieraus folgt:

Unabhängig vom theoretischen Maximum des verwendeten Netzwerkchips wird die maximale effektive Durchsatzrate eines IP-Stacks ohne Sliding Window durch die Round-Trip-Time und das verwendete Delayed Acknowledgment Timeout des beteiligten Kommunikationspartners begrenzt. Bei der Verwendung eines IP-Stacks ohne Sliding Window führt die Verwendung von Delayed Acknowledgement durch den Partner beim Datenversand von großen Datenmengen zu einem erheblich geringeren Datendurchsatz.

4 Anforderungsanalyse

Im folgenden Abschnitt werden die wesentlichen Anforderungen an eine Bibliothek für Webbrowser-basierte Bedienoberflächen von Mess- und Regelanwendungen auf Ethernetfähigen Mikroprozessorsystemen ermittelt. Der Übersicht halber sind die Anforderungen in verschiedene Anforderungskategorien unterteilt. Am Ende der jeweiligen Kategorie werden die konkreten Anforderungen aufgeführt. Dieses Kapiel dient nur dem Zweck, die grundsätzlichen Anforderungen festzulegen. Im Konzept in Kapitel 5 werden die erhobenen Anforderungen teilweise genauer ausgeführt, beziehungsweise erweitert.

4.1 Grundsätzliches

Wohldimensionierte Anwendungen im Mikroprozessor-Bereich werden häufig auf großzügigen Entwicklerplattformen entwickelt. Nachdem alle Bestandteile der Anwendung realisiert sind, wird der kleinstmögliche Prozessor der Produktfamilie ausgewählt, der im Stande ist, die Anwendung zu bewältigen. Damit die Verwendung der Bibliothek in diesem Sinne keine weiteren Kosten verursacht, ist strengstens darauf zu achten dass sie möglichst wenige Ressourcen verbraucht. Mikroprozessorsysteme verfügen meist über mehr Programmspeicher als Arbeitsspeicher. Eine Bibliothek hat eine unterstützende Funktion. Die Konsequenzen welche die Verwendung der Bibliothek mit sich bringt, sollten auf das mögliche Minimum reduziert werden. Die Bibliothek sollte der Anwendung so wenig spezielle Vorschriften machen wie möglich. Die Anwendungsschnittstelle sollte auf die genauen Anwendungsfälle zugeschnitten und möglichst einfach verwendbar sein.

Es sollte keine spezielle Software auf der Client-Plattform installiert werden müssen. Zu den verfügbaren Technologien auf modernen Systemen zählen die gängigen Browser-Plugins wie Flash und Java. Durch die vielen unterschiedlichen Browser und Plattformen entstehen sehr schnell Inkompatiblitäten. Außerdem variiert die Displaygröße auf unterschiedlichen Client-Plattformen. Es sollte keine spezielle Displaygröße vorgeschrieben werden. Auch kleinere Displays müssen optimal ausgenutzt werden können.

4.1.1 Resultierende Anforderungen

- **R1.1** - Die Bibliothek muß möglichst wenig Systemressourcen verbrauchen.
- **R1.2** - Es ist sparsam mit Arbeitsspeicher umzugehen.
- **R1.3** - Die Konsequenzen welche die Verwendung der Bibliothek mit sich bringt, sollten auf das mögliche Minimum reduziert werden.
- **R1.4** - Die Bibliothek muß einfach verwendbar sein.
- **R1.5** - Es müssen möglichst viele Browser unterstützt werden.
- **R1.6** - Die Displayfläche muß an die Eigenschaften des Clientsystems anpassbar sein.
- **R1.7** - Die Displayfläche muß optimal ausgenutzt werden.

4.2 Lose gekoppelte Oberflächen

4.2.1 Anwendungsfälle lose gekoppelter Oberfächen

Die ursprüngliche Motivaton für eine webbasierte Anwendungsoberfläche war die Verfügbarkeit von Rechenzeit und einem Display auf der Clientseite. Durch die Auslagerung der Anwendungsoberfläche auf einen externen Rechner lassen sich die Anwendungsfälle zunächst grob in drei verschiedene Kategorien aufspalten:

Kategorie A: Das System funktioniert auch ohne Benutzerinteraktion

Eine fortwährende Benutzerinteraktion ist für die Ausführung der Anwendung nicht notwendig. Der Benutzer verbindet sich nur temporär mit der Plattform, die Anwendung selbst funktioniert aber auch ohne Anwendungsoberfläche, da eine Benutzerinteraktion nicht zwingend notwendig ist. In dieser Kategorie könnte sich die Benutzerinteraktion zum Beispiel vorwiegend auf die Konfiguration und Justierung des Systems beschränken.

Kategorie B: Das System ist vorrübergehend auf Benutzerinteraktionen angewiesen

Eine fortwährende Benutzerinteraktion ist für die Ausführung der Anwendung nicht notwendig. Es gibt allerdings Momente, in denen eine Benutzerinteraktion zwingend erforderlich ist damit ein ordnungsgemäßer Betrieb weitergehen kann. Hat die Anwendung bereits ermitteln können, was die Ursache für die Ausnahmesituation ist, beschränkt sich die Benutzerinteraktion darauf, den Benutzer über die Fehlerquelle und mögliche Lösungsschritte zu informieren

und auf eine Bestätigung zu warten. Genauso könnte es aber auch sein, dass die Fehlerquelle nicht genau einzugrenzen ist. In dieser Kategorie könnte sich die Benutzerinteraktion zum Beispiel vorwiegend auf die Fehlerdiagnose und gelegentliche Wartung des Systems beschränken.

Kategorie C: Das System ist dauerhaft auf Benutzerinteraktion angewiesen

Für die Ausführung der Anwendung ist eine Benutzerschnittstelle dauerhaft notwendig, da die Anwendung nicht ohne regelmäßige Benutzereingaben funktioniert. In dieser Kategorie könnte sich die Benutzerinteraktion zum Beispiel vorwiegend auf die Darstellung prozessspezifischer Informationen und die Bereitstellung von Eingabemechanismen zur Versorgung des Systems mit Steuerdaten beschränken. Erwähnenswert bei dieser Kategorie ist, dass für die Dauer der Ausführung beide Rechner zwingend erforderlich sind.

4.2.2 Resultierende Anforderungen

- **R2.1** - Die Anwendung muß über Ereignisse wie das Verbinden, beziehungsweise Trennen der Oberfläche benachrichtigt werden. Es wird ein **Eventsystem** benötigt.
- **R2.2** - Das Eventsystem muß mindestens folgende Ereignisse signalisieren: Benutzer verbunden, Benutzer getrennt;

4.3 Grafische Benutzerinteraktion

4.3.1 Anwendungsfälle grafischer Benutzerinteraktion

Die grafische Benutzerinteraktion auf einem externen Gerät, das über einen Browser und eine Netzwerkschnittstelle verfügt, entlastet die CPU des Messsystems und bietet die Möglichkeit, unterschiedliche Clients mit unterschiedlichen Webbrowsern einzusetzen, die über unterschiedlich viel Displayfläche verfügen.

Anwendungsfall 1: Das Betrachten der wichtigsten Signale und Regelwerte mit ihren momentanen Werten

Ein typischer Anwendungsfall ist das Betrachten der wichtigsten Signale und Regelwerte mit ihrem momentanen Werten. Damit der Benutzer den aktuellen Zustand der Anwendung möglichst schnell erkennt, sind digitale oder analoge

Anzeigen, Balkendiagramme, Zeigerinstumente oder spezielle Anwendungsbezogene Instrumente wie Drehzahlmesser und Thermometer notwendig.

Anwendungsfall 2: Das Betrachten von Signalen und Regelwerten in einem zeitlichen Zusammenhang.

Ein typischer Anwendungsfall ist das Betrachten von Signalen und Regelwerten in einem zeitlichen Zusammenhang. Ein Vektor aller Signale und Regelwerte zu einem bestimmten Zeitpunkt beschreibt den aktuellen Regelzustand des Systems. Diese Daten können zur Funktionskontrolle, Fehlerdiagnose und Justierung des Systems dienen. Eine guter Vergleich hierzu ist die Oberfläche eines Oszilloskops. Hier werden die Pegel mehrerer Eingänge über die Zeit dargestellt. Um die Daten auswerten zu können, kann bei Oszilloskopen eine Zeitbasis gewählt werden, um größere und kleinere Zeitabschnitte auf der Darstellungsfläche darzustellen.

Anwendungsfall 3: Die gezielte Beeinflussung von Parametern

Ein weiterer Anwendungsfall ist die gezielte Beeinflussung von Parametern, welche das Verhalten des Systems direkt beeinflussen. Hierzu sind zunächst typische Steuerelemente von grafischen Benutzeroberfächen notwendig, wie zum Beispiel Knöpfe, Dreh- oder Schieberegler und Eingabefelder. Viele dieser Komponenten haben auch einen visuell beschreibenden Charakter. Zum Beispiel kann man an der Position eines Schiebereglers sehr schnell erkennen, wo sich der aktuelle Wert eines Reglers im Verhältnis zu seinem Regelspektrum befindet.

Anwendungsfall 4: Das gleichzeitige Betrachten von Signalen und gezielte Beeinflussen von Parametern

Oftmals resultiert das Anpassen von Parametern in einem geänderten Verhalten des Systems und somit auch seiner Prozessdaten. Oftmals werden Parameter händisch angepasst, bis ein gewünschter Wert erreicht ist. Ein Beispiel ist die Kalibration eines Sensors in einem System. Der Benutzer schließt ein externes Messgerät parallel zu einem der Sensoren des Systems an und liest den Wert ab. Dann vergleicht er die Messwerte des Systems mit denen des Messgeräts und passt die Parameter des Sensors an, bis die beiden Werte übereinstimmen.

Es wird ein Mechanismus benötigt, welcher die Ereignisse der Bedienelemente der Oberfläche an die Anwendung weitergibt. Die Kommunikation zwischen der Anwendung und der Bibliothek sollte möglichst einfach gestaltet sein. Da durch die aus Abschnitt 4.2 erhobenen Anforderungen bereits ein Eventsystem

benötigt wird ist es wünschenswert, dass die Ereignisse der Bedienelemente in dieses Eventsystem integriert werden.

Es wird ein Mechanismus benötigt, welcher die Bedienelemente der Oberfläche mit Zustandsinformationen versorgen kann. Es ist notwendig, dass der Zustand eines Bedienelements durch die Anwendung synchronisiert werden kann. Als Beispiel soll hier ein einrastender Knopf dienen. Es darf nicht vorkommen, dass ein Knopf nach einer Bedienung beispielsweise eine „Ich bin eingerastet“ - Nachricht versendet. Die Einrastposition dient in erster Linie dazu, dem Benutzer einen Anwendungszustand visuell darzustellen. Grundsätzlich sollte in so einem Fall eine „Ich werde gedrückt“ - Nachricht versendet werden, woraufhin die Antwort „Du bist eingerastet“ erfolgt.

4.3.2 Resultierende Anforderungen

- **R3.1** - Es muß möglich sein, mehrere Prozessparameter (Messdaten) in einem zeitlichen Zusammenhang zu betrachten.
- **R3.2** - Es werden Bedienelemente wie Knöpfe, Dreh- und Schieberegler, sowie Eingabefelder benötigt.
- **R3.3** - Bedienelemente müssen Steuerbefehle an die Anwendung senden können
- **R3.4** - Bedienelemente müssen Statusinformationen von der Anwendung erhalten können
- **R3.5** - Das Eventsystem muß Ereignisse der Bedienelemente der Oberfläche verarbeiten.
- **R3.6** - Es muß Mechanismen zur genauen Anpassung der Komponenten an die Bedürfnisse der Anwendung geben.
- **R3.7** - Es muß möglich sein, das System um anwendungsbezogene Anzeigeinstrumente zu erweitern.
- **R2.8** - Es muß möglich sein, gleichzeitig Bedienelemente zu verwenden und Prozessparameter zu betrachten.

4.4 Rollen (Mehrerere Benutzer)

4.4.1 Anwendungsfälle mehrerer Benutzer

Anwendungsfall 4: Die Bedienung eines Systems durch verschiedene Personen

Auf Mikroprozessorbasierten Sytemen wird typischerweise nur jeweils eine Anwendung ausgeführt. Bei einigen Systemen ist die Bedienung durch verschiedene Personen vorgesehen. Die Beeinflussung von bestimmten Anwendungsparametern ist unter Umständen nur mit genauen Kenntnissen des Systems möglich oder fällt in die Verantwortungsbereiche unterschiedlicher Personen.

Meist lassen sich die Anwendungsparamter zu einzelnen Bedienkontexten zusammenfassen. Um so kritischer die Beeinflussung eines Anwendungsparameters ist, desto höher ist die Verantwortung, die mit der Beinflussung dieses Parameters einher geht. Aus diesem Grund ist eine hierarchische Zugangsbeschränkung zu den unterschiedlichen Bedienungskontexten des Systems notwendig.

4.4.2 Resultierende Anforderungen

- **R4.1** - Es wird ein hierarchisches Zugangsberechtigungssystem für mehrere Benutzer benötigt.
- **R4.2** - Bedienelemente müssen durch das Zugangsberechtigungsystem abgesichert sein.

4.5 Erfassung von Messdaten

4.5.1 Übergabe von Messdaten an die Bibliothek

Damit die Anwendung Messdaten an die Bibliothek weitergeben kann, sind Mechanismen notwendig, um diese zu übergeben. Hierbei gibt es grundsätzlich zwei Vorgehensweisen:

Vorgehensweise A: Die Anwendung übergibt die Messdaten an die Bibliothek

In den meisten Mess- und Regelanwendungen gibt es regelmäßige Routinen, die für die Erfassung, Verarbeitung und Ausgabe von Mess- und Regelgrößen verantwortlich sind. Eine mögliche Vorgehensweise der Übergabe dieser Daten an die Bibliothek ist der einfache Aufruf einer Bibliotheksfunktion.

Vorgehensweise B: Die Bibliothek holt sich die Messdaten selbstständig von der Anwendung

Die regelmäßige Datenübergabe der Anwendung an die Bibliothek ist eine immer wiederkehrende Aufgabe. Die Aufrufe einer Funktion zur Übergabe der Daten an die Anwendung verstreuen sich zwangsläufig durch die gesamte Anwendung. Eine mögliche Vorgehensweise der Übergabe dieser Daten an die Bibliothek ist, der Bibliothek zu ermöglichen, Messdaten von der Anwendung abzuholen.

Vorgehensweise A ist eine sehr natürliche Herangehensweise, da es auch in anderen Bereichen der Datenkommunikation üblich ist, Senderoutinen zu verwenden, welche durch die Anwendung aufgerufen werden können. Diese Vorgehensweise bietet außerdem die Möglichkeit genau zu bestimmen, welche Messdaten zu welcher Zeit übertragen werden. Für Vorgehensweise B ist zwar ein Mechanismus bereitzustellen oder zu bedienen, welcher die Abholung von Messwerten durch die Bibliothek bereitstellt, allerdings kann bei dieser Vorgehensweise ein grundsätzlich anderes und in anderen Anwendungsbereichen teilweise sehr übliches Konzept realisiert werden. Die beiden Vorgehensweisen schließen sich nicht gegenseitig aus und sind grundsätzlich beide wünschenswert.

4.5.2 Erfassung von Messdaten durch die Bibliothek

Sobald es einen Mechanismus gibt, mit dem die Bibliothek Daten von der Anwendung abholt, muß auch festgelegt werden, wann diese Daten abgeholt

werden. Hier gibt es grundsätzlich zwei Ansätze:

Ansatz A: Der Clientrechner bekommt die Daten auf Anfrage geliefert

Bei diesem Ansatz ist ein Befehl im Protokoll notwendig, der den Mikroprozessor dazu veranlasst, einen Messwert aufzunehmen und diesen an die Oberfläche zu senden.

Ansatz B: Der Clientrechner bekommt die Daten automatisch in einer gewissen Frequenz geliefert

Bei diesem Ansatz übergibt der Mikrocontroller die Daten automatisch an den Client. Hierfür benötigt die Bibliothek einen eigenen Timer, damit Messwerte periodisch versendet werden können.

Bei Ansatz A entsteht ein erhöhtes Datenaufkommen im Netzwerk, da jedes Sample einzeln angefordert werden muß. Variable Netzwerklatenzzeiten können die Periodendauer der Messung ungenau machen. Ein Mechanismus zur Anforderung von Messdaten läßt sich außerdem über die Vorgehensweise A aus Abschnitt 4.5.1 selbst durch den Entwickler realisieren, falls dies von nöten sein sollte. Da durch die Forderungen aus Abschnitt 4.6 sowieso ein Timer notwendig ist, ist dieser Ansatz dagegen sehr vielversprechend.

4.5.3 Datentypen

Wie wir aus Abschnitt 3.2 wissen, sind die Einsatzgebiete von Mess- und Regelsystemen so vielfältig und unterschiedlich, wie es Regelaufgaben aus allen Bereichen des Haushaltes, der Industrie, der Luft- und Raumfahrt, Forschung usw. gibt. Hierdurch gibt es eine Vielzahl an Messgrößen in dieser Klasse von Anwendungen. Hierzu gehören sowohl klassische physikalische Größen wie Länge, Gewicht, Kraft, Geschwindigkeit, Druck, Spannung oder Strom, als auch sehr Anwendungsspezifische Größen, deren Anzahl schier unendlich ist. Diese Messgrößen werden von der Anwendung meist durch einen AD-Wandler, an dem ein Sensor angeschlossen ist, erfasst und durch die Auflösung des Wandlers in einen diskreten Wertebereich abgebildet. Die Auflösung dieser Werte ist von der Auflösung des Wandlers abhängig. Günstige Mikroprozessoren bieten oft schon AD-Wandler mit einer Auflösung von mindestens acht Bit. Bessere Modelle bieten Wandler mit zwölf oder mehr Bit. Externe AD-Wandlereinheiten hingegen bieten bereits Auflösungen von 16, 24 oder mehr Bit. Messwerte werden als diskrete Wertebereiche beliebiger (Bit)Breite verarbeitet. Da Mikroprozessoren diese Werte normalerweise nur in Einheiten von 2^{8*n} verarbeiten, werden diese Werte in den nächst größeren Datentyp abgespeichert.

AD-Wandlereinheiten verwenden zur Skalierung der Sensordaten auf einen Wertebereich die sogenannte *Referenzspannung*. Teilweise werden Sensoren elektrisch vorverarbeitet, welche durch Operationsverstärkerschaltungen, Spannungsteiler und Messbrücken geschieht. Einige Messwerte müssen durch die CPU nachverarbeitet werden, wobei unter der Kenntnis der Gegebenheiten der Sensor- und Wandleranordnung die Werte der AD-Wandlereinheit umgerechnet werden. Bei der Vereinbarung von Datentypen für die Oberfläche spielt die Art der Messgröße keine weitere Rolle. Es muß allenfalls für die Oberfläche eine menschenlesbare Form, Bezeichnung und Einheit angegeben werden.

Damit keine weitere Vorverarbeitung der Daten notwendig ist, sollte die Bibliothek vorzeichenbehaftete, wie auch nicht vorzeichenbehaftete Werte entgegennehmen und diese gegebenenfalls skalieren können.

Bis hier ist nur die Darstellung von Funktionsverläufen gefordert. Sobald die Oberfläche allerdings spezielle Messdaten darstellen möchte, für die eine besondere Darstellungsweise notwendig ist, muß es möglich sein, die Bibliothek um eigene Darstellungsweisen zu erweitern. Dies betrifft nicht die Übertragung der Daten im Protokoll. Aus diesem Grunde ist eine Datentyp-Vereinbarung für Messquellen erforderlich, durch die eine Anbindung selbst implementierter Darstellungsweisen vereinbart werden kann.

4.5.4 Resultierende Anforderungen

- **R5.1** - Es ist ein Mechanismus zur Übergabe von Messdaten an die Bibliothek notwendig (PUSH-Modus).
- **R5.2** - Es ist ein Mechanismus zur Bereitstellung von Messdaten notwendig, mit dem die Bibliothek Messdaten abholen kann (PULL-MODUS).

- **R5.3** - Die Bibliothek muß einen Datentypen für Messdaten bereitstellen, der die Bezeichnung und die Einheit in menschenlesbarer Form enthält.
- **R5.4** - Die Bibliothek muß im Stande sein, diskrete Werte beliebiger Breite mit und ohne Vorzeichen zu skalieren
- **R5.5** - Es ist eine Datentyp-Vereinbarung für Messquellen erforderlich, durch die eine Anbindung selbst implementierter Darstellungsweisen realisiert werden kann

4.6 Timing

Bei der Erfassung von Messdaten ist es notendig, den Zeitpunkt (Zeitstempel) der Erfassung aufzuzeichnen, um die Daten später in eine zeitliche Reihenfolge und miteinander in Zusammenhang bringen zu können. Hierbei sind grundsätzlich zwei verschiedene Vorgehensweisen denkbar:

4.6.1 Erzeugung von Zeitstempeln

Ansatz A: Die Zeitstempel werden beim Eintreffen auf dem Client erzeugt

Bei diesem Ansatz hilft die Tatsache, dass Desktoprechner und die meisten Clients, die über einen Browser verfügen, im Gegensatz zu einem Mikroprozessor meist auch über eine Uhr verfügen. Der Clientrechner könnte eintreffende Messdaten sofort mit einem Zeitstempel versehen. Etwaige Netzwerklatenzzeiten müssten hierbei allerdings berücksichtigt werden. Die Reihenfolge der Daten und somit die zeitliche Konsistenz wäre durch die Verwendung von TCP/IP sichergestellt. Allerdings läßt sich der genaue Zeitpunkt des Absendens nicht berechnen, da nur die *Round-Trip-Time*(RTT) ermittelt werden kann. Außerdem kann es passieren, dass Pakete neu übertragen werden müssen, was eventuell zu einer erheblichen Abweichung des Zeitstempels führen würde.

Ansatz B: Die Zeitstempel werden auf dem Mikroprozessor erzeugt

Bei diesem Ansatz werden die Zeitstempel auf dem Mikroprozessor erzeugt und zusammen mit dem Messwert an den Client übertragen. Hierfür benötigt der Mikroprozessor eine Uhr. Etwaige Netzwerklatenzzeiten spielen bei diesem Ansatz keine Rolle.

Ansatz A kommt durch die beschriebenen Problematiken nicht in Frage. Aus Ansatz B resultiert zwar ein höheres Datenvolumen bei der Übertragung von Messwerten, doch um präzise Zeitstempel zu bekommen müssen die Zeitstempel auf dem Mikroprozessor erzeugt werden. Hierfür benötigt die Bibliothek einen Timer.

Üblicherweise werden Zeitstempel als Sekunden, Millisekunden oder Mikrosekunden seit 1970 gezählt und stellen einen absoluten Zeitpunkt dar. Dieses Format hat den Vorteil, dass eine Zeitspanne zwischen zwei Zeitstempeln sehr schnell berechnet werden kann, indem man einfach die Differenz bildet. Die meisten Computersysteme verwenden intern eines dieser Formate.

Ein Zeitstempelformat auf Millisekunden-Basis ist nicht hoch genug aufgelöst, da bei Anwendungen auf Mikrocontrollern teilweise auch Zeitspannen unter einer Millisekunde eine Rolle spielen. Eine Millisekunde ist eine Zeitspanne, in der ein Prozessor mit 1 MHz bereits 1000 Taktzyklen ausführen kann. Damit die Zeitstempel auch höchsten Ansprüchen genügen, wäre ein Zeitstempelformat notwendig, welches Mikrosekunden abbildet. Hier ist allerdings abzuwägen, welche Genauigkeit auf dem System überhaupt machbar beziehungsweise notwendig ist. Ein Datumsformat in Mikrosekunden bietet den Vorteil, dass der Timer intern beispielsweise auch mit einer Periode von 500 Mikrosekun-

den (1/2 Millisekunde) oder 100 Mikrosekunden (1/10 Millisekunde) arbeiten kann. Dies würde zwar die Genauigkeit des Zeitstempels nicht voll ausschöpfen, doch das System wird skalierbar, was die zeitliche Auflösung betrifft.

4.6.2 Resultierende Anforderungen

- **R6.1** - Die Bibliothek benötigt eine Uhr (Timer) um Zeitstempel zu generieren.
- **R6.2** - Das Zeitstempelformat muß Mikrosekunden auflösen können
- **R6.3** - Das Nachrichtenformat für die Übertragung von Messwerten muß ein Feld für einen Zeitstempel vorsehen.

4.7 Protokoll

Bei der Kommunikation mit der Benutzeroberfläche ist ein Protokoll notwendig. Es ist notwendig, das Format, den Inhalt, die Bedeutung und die Reihenfolge der gesendeten Nachrichten zwischen dem Mikroprozessor und der Oberfläche festzulegen. Da die Ressourcen auf einem Mikroprozessor ein knappes Gut sind, ist ein Protokoll notwendig, welches für den Mikroprozessor möglichst einfach zu verarbeiten ist. Für die Realisierung eines Protokolls wird zunächst ein Transportprotokoll auf der IP-Schicht benötigt. Hier kommen vor allem die am weitesten verbreiteten Transportprotokolle in Frage, welche auf der IP-Schicht zur Verfügung stehen. Die gängigsten Transportprotokolle sind TCP[1] und UDP[2].

Die beiden Transportprotokolle sind in Abschnitt 3.3 bereits beschrieben worden.

Egal, welche Technik letztendlich für die Realisation der Bibliothek verwendet wird - für die Auslieferung einer initialen Webseite per HTTP wird der Mikroprozessor auf jeden Fall TCP benötigen. Dennoch gibt es Webtechnologien wie zum Beispiel Java-Applets, welche eine UDP-Verbindung zum Mikroprozessor öffnen könnten. Es gilt abzuwägen, welches Transportprotokoll gewählt werden sollte. Die Verwendung von UDP ist durch den nicht benötigten Speicher für Neuübertragungen ressourcenschonender als TCP. Kleinere Datenverluste können bei Anwendungen wie dem Streaming von MP3 oder Mpeg oftmals ignoriert werden, sie führen allenfalls zu Bildartefakten, Rucklern und kleinen Störungen. Bei einem Anwendungsprotokoll das auf UDP aufsetzt, ist ein Datenverlust dagegen kritisch. Die Anwendung wird selbst Mechanismen bereitstellen müssen, um verlorengegangene Daten neu zu übertragen. Die Vorteile der Speicherersparnis bei der Verwendung von UDP werden hierdurch stark

[1] *Transmission Control Protocol*
[2] *User Datagram Protocol*

relativiert. TCP/IP ist das zugrundeliegende Transportprotokoll für die Kommunikation mit Webservern, die Übertragung von E-Mail, FTP, Telnet, und vielen anderen bekannten Anwendungen.

Da es sich bei der Kommunikation zwischen der Oberfläche und dem Mikroprozessorsystem um eine Punkt-zu-Punkt Verbindung handelt, soll das fehlerredundantere TCP Transportprotokoll zur Verwendung kommen.

4.7.1 Resultierende Anforderungen

- **R7.1** - Es ist ein Protokoll zu entwickeln, welches für den Mikroprozessor möglichst einfach zu verarbeiten ist.
- **R7.2** - Als Transportprotokoll ist TCP zu wählen
- **R7.3** - Das Nachrichtenformat, die Bedeutung und die Reihenfolge der Nachrichten ist genau zu spezifizieren.

4.8 IP-Stack

Für den Betrieb eines Netzwerkcontrollers ist ein IP-Stack notwendig. Da die Bibliothek ressourcenschonend sein soll ist hierbei im speziellen darauf zu achten, dass ein Stack gewählt wird, der möglichst ressourcenschonend arbeitet. (Siehe Grundlagenkapitel 3.3.2)

Die eigentliche Aufgabe des IP-Stacks ist, die Datentransfers zwischen der Mikroprozessorplattform und dem Client-Rechner zu verarbeiten. Der IP-Stack muß im Stande sein, das hierbei aufkommende Datenvolumen zu übertragen.

4.8.1 Resultierende Anforderungen

- **R8.1** - Es ist ein IP-Stack auszuwählen, der TCP beherrscht(Siehe **R7.2**)

- **R8.2** - Der IP-Stack muß möglichst ressourcenschonend arbeiten
- **R8.3** - Der Stack muß das aufkommende Datenvolumen der Anwendung bewältigen können

5 Konzept

5.0.2 Funktionale Komponenten des Konzepts

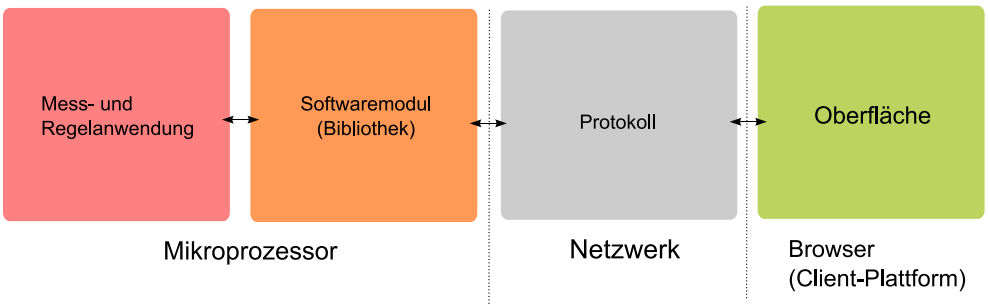


Abbildung 5.1: Übersicht der Komponenten

Ein wichtiger Bestandteil der Bibliothek ist ein Softwaremodul, welches eine Anwendungsschnittstelle für die eigentliche Anwendung auf dem Mikroprozessor bereitstellt. Das Softwaremodul hat Mechanismen bereitzustellen, welche auf der einen Seite die nötigen Bibliotheksfunktionen für die Verarbeitung von Messwerten, sowie teilweise Steuerung der Anwendungsoberfläche im Browser und die Weitergabe von Steuerbefehlen der Oberfläche an die Anwendung enthalten. Auf der anderen Seite bedient die Anwendung die Schnittstelle des IP-Stacks, um die Kommunikation mit dem Netzwerk abzuwickeln, welches den Mikroprozessor mit der Oberfläche (im Folgenden teilweise auch Client genannt) verbindet.

Für die Kommunikation mit der Oberfläche ist ein Protokoll notwendig, welches das Format, den Inhalt, die Bedeutung und die Reihenfolge der gesendeten Nachrichten zwischen dem Mikroprozessor und der Oberfläche festlegt. Dieses Protokoll sollte für die Verarbeitung durch den Mikroprozessor optimiert sein, da die Ressourcen auf der Mikroprozessorplattform mit der Anwendung geteilt werden müssen.

Der letzte wichtige Bestandteil der Bibliothek ist die Oberfläche. Die Oberfläche muß Mechanismen zur Steuerung der Anwendung, sowie Komponenten zur Darstellung von Mess- und Prozessdaten bereitstellen. Hierzu kommen unterschiedliche Webtechnologien in Frage. Da diese einen großen Einfluß auf das Protokoll und die Struktur der Bibliothek hat, wird zunächst ein Konzept für die grafische Oberfläche entwickelt.

5.1 Oberflächenkonzept

Die Oberfläche ist das Gesicht der Mess- und Regelanwendung (Im weiteren einfach Anwendung genannt). Es gibt viele unterschiedliche Auffassungen darüber, wie eine grafische Oberfläche aussehen soll. Bei der Entwicklung eines Oberflächenkonzepts können an dieser Stelle sicher nicht alle glücklich gemacht werden. Viele Anwendungen, zum Beispiel im Multimedia-Bereich, verwenden avantgardistische Layouts, die eher verspielt als anwendungsorientiert sind. Oberflächendesign spielt eine große Rolle, wenn eine gewisse Zielgruppe angesprochen werden soll. Hier sei als Beispiel das grafische Konzept von Mac OS X angeführt. Die Corporate Identity findet sich vom IPod über Itunes bis hin zum Betriebssystem an jeder Stelle wieder. Eine Oberflächenbibliothek zu entwickeln, welche alle Möglichkeiten des Layouts bietet wie zum Beispiel GTK, QT oder Swing, ist sehr komplex. Die Entwicklung einer Bibliothek die auch nur annähernd die Funktionalität bietet, welche in solchen Bibliotheken vorhanden ist, wäre zwar wünschenswert, würde jedoch den zeitlichen Rahmen dieser Arbeit und die Anforderungen an das Anwendungsfeld weit überschreiten.

Die Oberfläche der Bibliothek ist die Oberfläche einer technischen Anwendung. Hier ist vor allem die Erfüllung aller Anforderungen und der Entwurf eines schlüssigen Konzepts gefragt.

Zunächst gilt es, den Anforderungen an die Oberfläche eine Form zu geben.

Aus den Anforderungen in Abschnitt 4.3.2 geht hervor, dass die gleichzeitige Betrachtung von Messwerten bzw. Prozessparametern in einem zeitlichen Zusammenhang notwendig ist. Die Oberfläche eines Oszilloskops dient hier als Beispiel. Bei einem Oszilloskop werden die Pegel mehrerer Eingänge über die Zeit dargestellt. Um die Daten auswerten zu können ist es bei Oszilloskopen möglich, eine Zeitbasis zu wählen, um größere und kleinere Zeitabschnitte auf der Darstellungsfläche anzuzeigen. Die Oberfläche eines Oszilloskops scheint sich bewährt zu haben, denn auch bei modernen digitalen Speicheroszilloskopen wird dieses Konzept weitergeführt, obwohl im Inneren dieser Geräte ein Rechner arbeitet, welcher durchaus andere Darstellungsarten bieten könnte.

Ein Konzept, bei dem feste Displaybereiche für konkrete Messquellen vorgesehen werden können, ist stark von der Displayfläche des Clients abhängig und in dieser Hinsicht schlecht skalierbar. Aus diesem Grunde ist es sinnvoller, stattdessen eine einzige Displayfläche vorzusehen, und dem Anwender die Wahl zu überlassen, welche Datenreihen er in diesem Bereich gerade betrachten möchte. Eine grafische Darstellung einer oder mehrerer Messreihen untereinander läßt sich gut skalieren. Damit das Augenmerk des Anwenders trotz dieser Wahlfreiheit auf die wichtigsten oder momentan kritischen Parameter der Anwendung gelenkt werden kann, kann das System dem Anwender eine sinvolle Vorauswahl präsentieren, welche bei der Verbindungsaufnahme mit dem System automatisch dargestellt wird. Hierfür ist im Konzept ein Be-

fehl vorgesehen, mit dem die Anwendung zu jeder Zeit beliebige registrierte Messquellen auf der Displayfläche aktivieren kann. Da die Anwendung signalisiert bekommt, welcher Benutzer sich gerade verbunden hat, kann sie auch entscheiden, welche Vorauswahl der Benutzer präsentiert bekommt.

Eine weitere Anforderung an die Oberfläche ist, dass Bedienelemente wie Knöpfe, Dreh- und Schieberegler und weitere anwendungsbezogene Bedienungskomponenten benötigt werden. Damit diese Bedienelemente nicht einfach ungeordnet „herumfliegen“ ist eine Gruppierung dieser Displayelemente notwendig. Ein allgemeiner und oft gewählter Ansatz für die Positionierung von Komponenten ist ein freies Layout mit absoluten oder relativen Koordinaten auf der Displayfläche. Zunächst muß allerdings etwas über Bedienkontexte (im Weiteren auch Anwendungskontext oder einfach nur Kontext genannt) gesagt werden.

Die Notwendigkeit von Anwendungskontexten resultiert aus der Notwendigkeit, verschiedenen Benutzern mit unterschiedlichen Berechtigungsgraden nur die jeweils für ihren Berechtigungsgrad freigeschalteten Kontrolloberflächen zu bieten. In einem Anwendungskontext wird eine spezielle Sicht auf die Anwendung dargestellt, die einem Anwendungsfall entspricht. Die Bibliothek stellt also nicht eine einzige Oberfläche dar, sie muß in diesem Sinne mehrere Oberflächen darstellen. Ein Anwender könnte beispielsweise Zugang zu mehreren Anwendungskontexten haben. Hier kommt wieder die Ausnutzung der Displayfläche ins Spiel. Die gleichzeitige Darstellung aller verfügbaren Anwendungskontexte ist nicht nur schwer realisierbar, sie ist auch überhaupt nicht notwendig, da ein Anwendunskontext alle Bedienelemente für die Verarbeitung eines Anwendungsfalls enthalten kann. Indem man einen Displaybereich vorsieht, der einen Anwendungskontext darstellen kann und dem Benutzer die Wahlfreiheit überlässt, welchen der für ihn freigeschalteten Bedienkontexte er betrachten möchte, kann man analog zu der Auswahl der Messquellen vorgehen, das heißt einen Befehl vorsehen, mit dem die Anwendung dem Anwender einen Bedienkontext auf der Displayfläche aktivieren kann. Durch dieses Konzept wäre beispielsweise ein Szenario vorstellbar, in dem die Anwendung, die mit dieser Bibliothek erweitert wurde feststellt, dass ein kritischer Prozessparameter absolut aus dem Ruder läuft. Um dem Benutzer in dieser Situation zu assistieren, könnte sie die Datenreihe des Prozessparameters und den Bedienkontext zum Abbruch des Prozesses aktivieren.

An dieser Stelle stehen alle notwendigen Elemente der Oberfläche fest. Es kann ein erster Layoutversuch gewagt werden. Abbildung 5.30 zeigt ein Darstellungskonzept, welches die bisher beschriebenen Elemente miteinander vereint.

Die Grundkonfiguration der Bibliothek sollte allerdings auch vorsehen, die einzelnen Bereiche global an- und auszuschalten, falls in einzelnen Projekten die eine oder andere Komponente nicht gebraucht wird. Besteht beispielsweise nur ein Kontext, und die obere Displayhälfte ist global deaktiviert, kann die untere Displayhälfte auch allein im klassischen Stil einer frei arrangierbaren

Oberfläche eingesetzt werden.

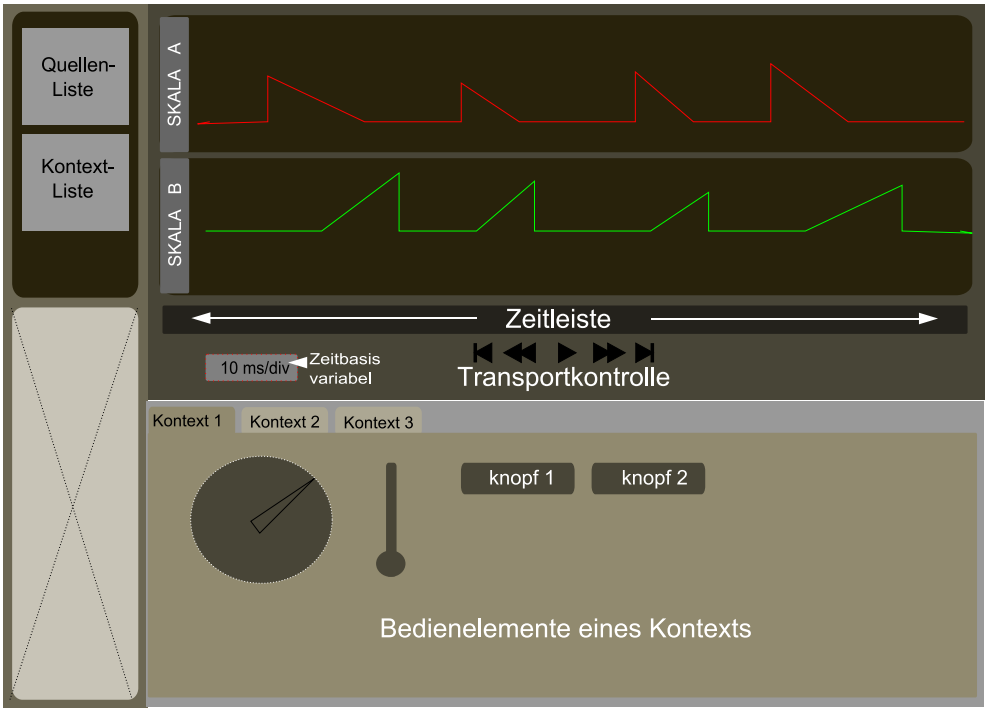


Abbildung 5.2: Entwurf der Oberfläche

Der in Abbildung 5.30 gezeigte Layoutentwurf zeigt eine mögliche Displayaufteilung, wie sie sich aus dem soeben aufgestellten Konzept ergibt. Das Layout basiert auf der einfachen Aufteilung der Displayfläche in drei Grundbereiche. Im oberen Displaybereich befindet sich ein Log-Viewer. Die Komponente ist weitestgehend der Oberfläche eines Oszilloskops nachempfunden. Sie unterscheidet sich allerdings von einem Oszilloskop, da die Datenreihen über eigene Skalen verfügen. Dies resultiert aus der Tatsache, dass in einer Messanwendung nicht nur Spannungen miteinander verglichen werden. Die Datenreihen haben jedoch eine gemeinsame Zeitbasis. Es ist also nicht notwendig, dass jede Datenreihe eine eigene Zeitleiste darstellt. Alle zur Darstellung aktivierten Messquellen werden innerhalb dieses Bereiches untereinander aufgelistet. Damit diese Displayfläche nicht zu klein für die gewählten Quellen wird, könnte ein Maximum von beispielsweise vier Datenquellen festgelegt werden, damit die Darstellung nicht zu klein wird. Es könnte aber auch ein Mechanismus zum Umschalten zwischen den aktivierten Datenreihen festgelegt werden. Über die Transportkontrolle kann das Zeitfenster des Log-Viewers verschoben werden.

Im unteren Displaybereich befindet sich die Darstellungsfläche für Bedienungskontexte, welche die eigentlichen Bedienelemente enthalten. Zwischen den aktivierten Bedienkontexten kann über Tabs umgeschaltet werden. Im linken Bereich befindet sich eine Listenkomponente mit verfügbaren Messquellen und Kontexten. Das gezeigte Layout macht einen ordentlichen Eindruck und erfüllt alle erhobenen Anforderungen.

5.1.1 Realisierungsansätze für die Oberfläche

Das in Abschnitt 5.1 gezeigte Layout basiert auf der Aufteilung der Displayfläche in drei Grundbereiche. Es werden Techniken zum Zeichnen von Funktionsplots, Tabs und möglichst viele Gui-Elemente benötigt. Alle im Folgenden vorgestellten Realisierungsansätze haben grundsätzlch das Potenzial, eine solche Oberfläche darzustellen. Bei der Auswahl eines geeigneten Ansatzes gibt es einige Dinge zu berücksichtigen:

- Welche Last wird auf dem Mikroprozessor erzeugt?
- Wie hoch ist die Komplexität des Ansatzes?
- Wie einfach ist der Ansatz erweiterbar?
- Ist ein Dateizugriff möglich?
- Wird ein Browserplugin benötigt?
- Ist zum Erstellen (und Erweitern) der Software eine Lizenz nötig?

Ajax (HTML/Javascript)

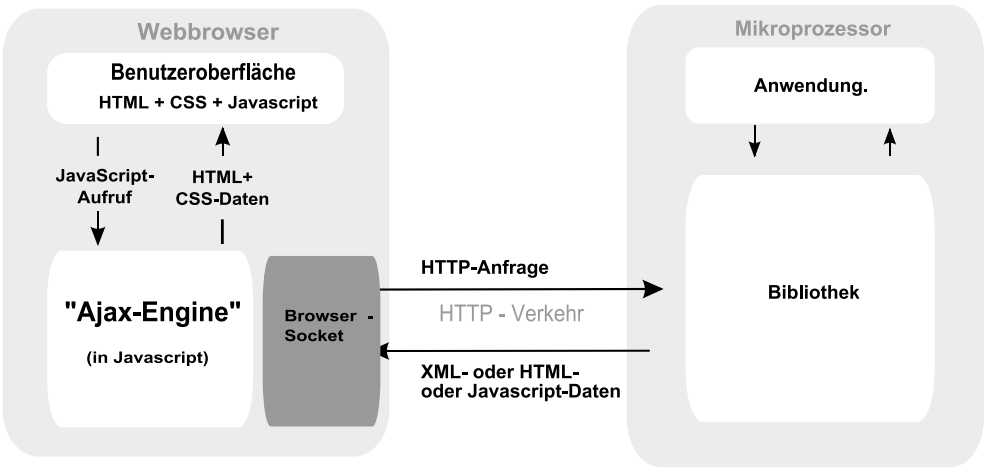


Abbildung 5.3: Ajax-Modell

HTML wurde ursprünglich entwickelt, um Textinformationen formatiert zu präsentieren, und ist allein nicht zur Entwicklung von Anwendungen geeignet. Ursprünglich gab es nur Links und Formulare zur Interaktion. Um eine dynamische Client-Anwendung im Browser zu realisieren, ist eine Programmiersprache notwendig. Mit Javascript steht jedem modernen Browser eine Programmiersprache zur Verfügung, die direkt auf die Anforderungen der dynamischen Verarbeitung von Webseiten in HTML zugeschnitten ist. Ajax bedeutet ***A****synchronous* ***J****avascript* ***a****nd* ***X****ML*. Es bezeichnet ein Konzept der Asynchronen Datenübertragung zwischen einem Server und dem Browser, das es ermöglicht, innerhalb einer HTML-Seite eine HTTP-Anfrage durchzuführen, ohne die Seite komplett neu laden zu müssen. Einzelne Abschnitte der Seite oder Nutzdaten können sukzessive nachgeladen werden. Hierzu wird

das XMLHttpRequest[1]-Objekt des Browsers verwendet. Dieses Objekt stellt jeder moderne Javascript-fähige Browser der Javascript-Umgebung zur Verfügung. Leider sind die verschiedenen XMLHttpRequest-Implementierungen nicht vollständig zueinander kompatibel. Ajax-Bibliotheken versuchen, diese Inkompatiblitäten zu verdecken. Sie verfügen intern über Mechanismen zur Erkennung des Browsers und stellen eine eigene Schnittstelle bereit, um mit dem XMLHttpRequest-Objekt zu arbeiten. Diese Schnittstelle ermöglicht es dem Benutzer oftmals, die asynchron eintreffenden Daten direkt im Dokumentenobjektmodell (DOM) abzulegen. Hierzu sind die Daten meist mit XML formatiert.

Bei einer Implementierung der Oberfläche im Ajax-Ansatz[2] wird zunächst eine Javascript-Engine übertragen. Der Browser aktiviert diese Engine nachdem er sie geladen hat. Die interaktiven Elemente der ausgelieferten Webseite wie Knöpfe, Textfelder und Ausgabebereiche sind mit dieser Ajax-Bibliothek verzahnt. Die Kommunikation mit dem Server wird per XML erledigt. Es gibt verschiedene grundlegende Ajax-Bibliotheken, um die Kommunikation abzuwickeln. Einige der bekanntesten sind *Prototype*[3], *Scriptaculous*[4] und *Rico*[5], wobei Scriptaculous bereits Visuelle Effekte (Ein- und Ausblenden etc.) und einige andere Mechanismen bietet, um interaktive grafische Benutzeroberflächen zu entwickeln. Inzwischen gibt es sogar Oberflächen-APIs wie zum Beispiel *ExtJS*[6], die einer vollwertigen Oberflächen-API sehr nahe kommen. Mit ExtJs ist es möglich, so gut wie keine einzige Zeile HTML zu schreiben. Es muß nur die Bibliothek ausgeliefert werden, jegliches Anwendungsverhalten kann dann in Javascript implementiert werden. Die Bibliothek verfügt über eine beträchtliche Auswahl an UI-Widgets, es sind sogar verschiedene LayoutManager und ein eigenes EventSystem vorhanden. Die Bibliothek gilt als ausgereift und stabil. In der ExtJs-Api[7] gibt es umfangreiche Funktionen zur Anwendungsentwicklung. ExtJS kümmert sich nicht nur um die Darstellung, sondern bietet komplette Klassen für Eventhandling, asynchrone Kommunikation, die Speicherung der Zustände der Widgets oder eigener Zustandsdaten als Cookies, komplexere Datenstrukturen, den DOM-Zugriff, JSON etc. Zur Darstellung von Graphen/Histogrammen etc. ist ExtJs nicht geeignet. Hierzu bietet sich etwa der Charting-Part der Javascript-Bibliothek *dojo*[8] an.

Der dynamische Zugriff auf das Dateisystem ist mit Javascript nicht möglich. Es ist allenfalls möglich, eine Datei zum manuellen Herunterladen durch den Anwender bereitzustellen.

Webserver sind nicht in der Lage, asynchrone Events an einen Ajax-Client auszuliefern. Um Daten zu empfangen, muss der Ajax-Client seinerseits per-

[1] *Wikipedia* `http://de.wikipedia.org/wiki/XMLHttpRequest`
[2] *Wikipedia* `http://de.wikipedia.org/wiki/Ajax_(Programmierung)`
[3] `http://www.prototypejs.org/`
[4] `http://script.aculo.us/`
[5] `http://openrico.org`
[6] `http://extjs.org`
[7] `http://extjs.com/deploy/dev/docs/`
[8] `http://dojocampus.org/explorer/#Dojox_Charting`

manent beim Webserver nachfragen (Polling), ob ein neues Ereignis stattgefunden hat, bzw. ob eine Änderung im Anwendungszustand stattgefunden hat. Dies erzeugt eine stetige Last auf dem Webserver, auch wenn keine Ereignisse stattfinden. Um ein andauerndes Polling zu vermeiden, wurde die Technik entwickelt, Poll-Anfragen solange zurückzuhalten, bis ein tatsächliches Ereignis oder ein Timeout eintritt. Eine Ajax-Anwendung ist im Idealfall serverseitig mit einer Anfrage verknüpft, die dazu benutzt werden kann, dem Anwendungs-Client beim Eintreten eines entsprechenden Events eine Antwort zu schicken.

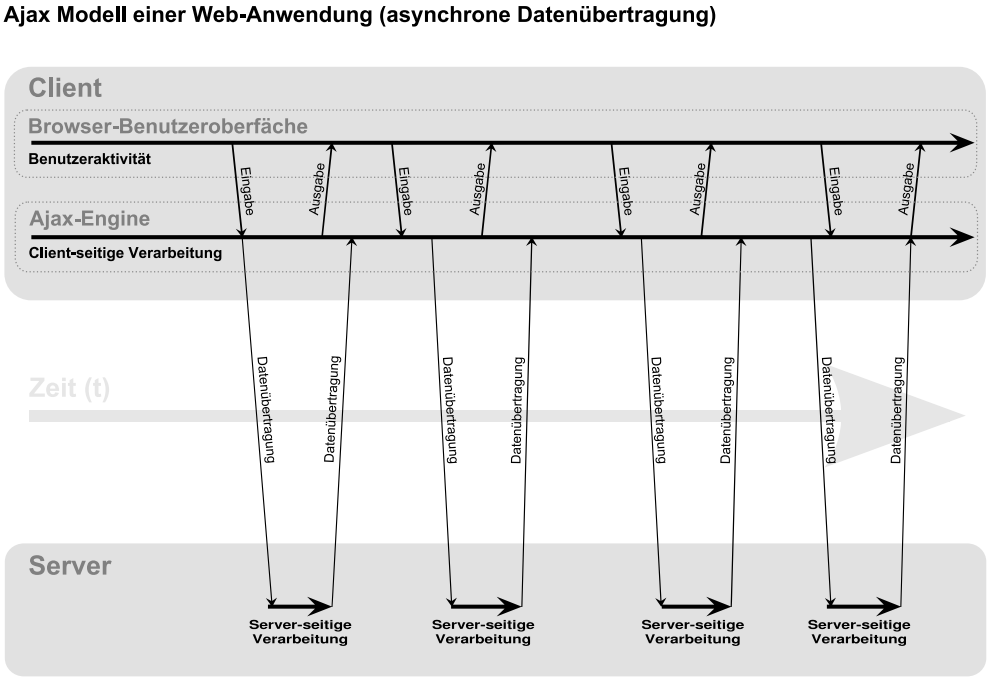


Abbildung 5.4: Ajax: Asynchrone Datenübertragung
Quelle: *Wikipedia*

Diese Technik wirft allerdings neue Probleme auf. Bisher war es üblich, pro Anfrage an den Server einen Thread zu erzeugen, dessen Ressource sofort nach dem Abarbeiten der Anfrage wieder freigegeben werden konnten. Bei der beschriebenen Polling-Technik ist diese Freigabe des Threads jedoch nicht möglich. Es bleiben also weiterhin Ressourcen, wie beispielsweise Speicher, belegt. Dieses Problem stellt neue Anforderungen an die Skalierbarkeit einer Ajax-Anwendung.

Flash™(Flex™/Air™)

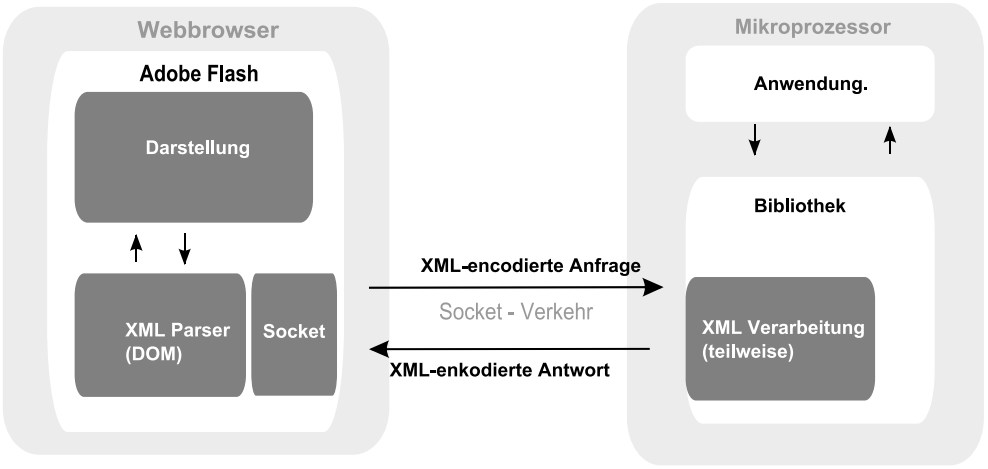


Abbildung 5.5: Flash-Modell

Flash™ist eine Technologie von Adobe®zur Erstellung von multimedialen Inhalten, sogenannter Flash-Filme. Flash-Filme werden mit der kommerzielle integrierten Entwicklungsumgebung Adobe Flash entwickelt. Zur Darstellung wird das Flash-Browserplugin benötigt. Als Programmiersprache dient ActionScript, ein Dialekt von ECMAScript[14], wodurch Code in Flash-Filmen grundsätzlich die gleiche Syntax hat, wie Javascript. Flash könnte auch benutzt werden, um den Ajax-Ansatz zum Beispiel bei den Graphen zu ergänzen. Hierfür können Flash-Filme bei der Einbettung in die Webseite parametrisiert werden, zum Beispiel mit den Daten für einen Funktionsplot. Homogene Lösungen sind allerdings eher anzustreben, damit die Erweiterbarkeit der Oberflächenbibliothek auch einfach zu realisieren ist. Flash wurde ursprünglich als reines Animationswerkzeug entwickelt und ist zunächst häufig nur verwendet worden, um animierte Banner in Webseiten zu erstellen. Seitdem Flash ab Version 4.0 allerdings um ActionScript erweitert wurde, kann es auch zur Erstellung von komplexen browserbasierten Anwendungen verwendet werden. Zur Kommunikation mit dem Server bietet Flash das XMLSocket Objekt[9]. Eine herkömmliche HTTP-Lösung fragt den Server in kurzen Abständen nach neuen Nachrichten ab und lädt diese mithilfe einer HTTP-Anforderung. Das XMLSocket Objekt verwendet anstatt der Sockets des Browsers seinen eigenen Socket zur Verbindung mit dem Server. Eine XMLSocket-basierte Kommunikation unterhält eine offene Verbindung zum Server, damit dieser neu eingegangene Nachrichten sofort und ohne Anforderung vom Client senden kann. Um die XMLSocket-Klasse verwenden zu können, muss auf dem Servercomputer ein Daemon ausgeführt werden, der das von der XMLSocket-Klasse verwendete Protokoll verarbeiten kann. Der Daemon verzichtet bei der Kommunikation auf HTTP. Die Kommunikation über diese Verbindung erfolgt in einem speziellen XML-Protokoll.

[9] `http://livedocs.adobe.com/flash/9.0_de/ActionScriptLangRefV3/flash/net/XMLSocket.html`

Um Flash-Anwendungen zu erzeugen, kann Adobe Flash verwendet werden, eine Entwicklungsumgebung die Werkzeuge zum Zeichnen und eine Zeitleiste beinhaltet, um Animationen zu entwickeln. Programmierer arbeiten allerdings nicht so gern mit den Zeichenwerkzeugen, Einstellungspanels und der Zeitleiste, um Oberflächen für Anwendungen zu entwickeln. Aus diesem Grunde wurde Adobe Flex entwickelt. Flex stellt dem Entwickler ein Framework von erweiterbaren Klassen und die Entwicklungsumgebung Flex Builder zur Verfügung, mit der grafische Anwendungsoberflächen mit MXML beschrieben werden können, die Logik wird weiterhin mit ActionScript programmiert. Flash und Flex lassen sich gut miteinander kombinieren, da Flex Programme als Flash-Film übersetzt werden. Hierbei kann Flash ergänzend verwendet werden, um beispielsweise Graphen zu zeichnen. Flex liefert allerdings auch eigene (teilweise optionale[10]) Komponenten, um Graphen zu zeichnen. Es ist auch ein kostenloses Flex SDK von Adobe verfügbar, welches einen Compiler enthält um Flash-Filme zu erzeugen. Das Flex SDK kommt mit einem Satz von Widgets, inklusive typischer GUI-Komponenten und Layoutmanager, die zur Erstellung von grafischen Oberfächen benötigt werden.

Flash kann bedingt auf die Festplatte des Clientrechners zugreifen, allerdings nur in einen durch das Flash-Plugin verwalteten Bereich. Dieser Bereich ist eher dafür vorgesehen, kleine Anwendungsbzeogene Daten zu speichern und ist deswegen nur bedingt für die Speicherung von Daten verwendbar[11].

Adobe® Air™ ist eine Laufzeitumgebung, um Web-Technologien für die Entwicklung plattformübergreifender Rich-Internet-Anwendungen für den Desktop zu entwickeln. Air kombiniert die Vorteile des Browsers mit der Funktionalität des Desktops. Anwendungen können in zwei Versionen bereitgestellt werden: Eine Browser-basierte Version für alle Anwender und eine Desktop-Version für die Verwendung auf dem Desktop.

[10] `http://livedocs.adobe.com/flex/201/langref/mx/charts/package-detail.html`
[11] `http://www.macromedia.com/support/documentation/de/flashplayer/help/help02.html`

Java™(Applet/Standalone)

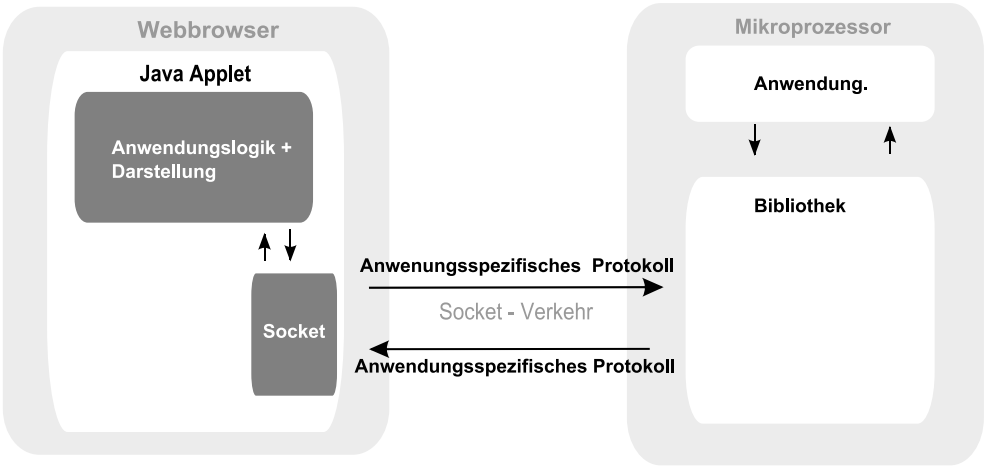


Abbildung 5.6: Java-Modell

Java™ ist eine objektorientierte Programmiersprache von Sun Microsystems®. Java Programme werden in sogenannten Bytecode übersetzt, der dann in einer virtuellen Maschine, der Java-Laufzeitumgebung, ausgeführt wird. Die Laufzeitumgebung ist für viele Betriebssysteme und Plattformen verfügbar, wodurch Java-Programme nur ein einziges Mal übersetzt werden müssen, und dann auf einer Vielzahl von Plattformen laufen. Java bietet außerdem Java Applets. Applets sind Java Programme, die sich in Webseiten einbetten lassen. Hierzu benötigt der Browser das Java-Plugin. Applets können das gesamte Spektrum der umfangreichen Java Klassenbibliothek verwenden. Die einzige Einschränkung hierbei ist, dass Applets nur Verbingung zu dem Host aufnehmen können, von dem sie geladen wurden und dass sie keinen Zugriff auf die lokale Festplatte haben. Möchte ein Applet dennoch diese Grenzen überschreiten, ist es notwendig, das Applet digital zu signieren. Signierte Applets müssen zunächst vom Anwender als vertrauenswürdig eingestuft werden, bevor sie ausgeführt werden können. Hierzu erscheint eine Dialogbox, die Aufschluss über den Herausgeber der Signatur und somit den Ursprung des Programms gibt. Danach können sich Applets verhalten, wie vollwertige Java-Anwendungen. Die Java-Klassenbibliothek enthält mit dem *Abstract Windowing Toolkit*(AWT) und *Swing* gleich zwei (wenn auch verwandte) Bibliotheken für die Erstellung von Grafischen Oberflächen. Mit Java2D existiert eine vollständige 2D-Grafikumgebung. Java2D wird häufig verwendet, um zusammen mit AWT oder Swing komplexe Oberflächen zu erstellen. Java Applets können ebenso wie Flash-Filme eigene Sockets aufmachen, um Daten mit dem Server auszutauschen, von dem sie geladen wurden. Java hat dabei den vollen Zugriff auf den Socket, eine Implementierung eines Mikroprozessorfreundlichen Protokolls ist somit möglich. Mit der Netbeans IDE und Eclipse stehen zwei mächtige Entwicklungsumgebungen zur Verfügung, die kostenlos erhältlich sind.

5.1.2 Bewertung

Welche Last wird auf dem Mikroprozessor erzeugt?

Die verschiedenen Ansätze verlangen dem Mikroprozessor unterschiedlich viel Prozessorlast ab. Dies hat damit zu tun, dass die Ansätze unterschiedlich viel mit dem Mikroprozessorsystem kommunizieren müssen und dass dabei teilweise spezielle Kodierungen notwendig sind.

- Ajax (HTML/Javascript)
  Durch die Notwendigkeit der Verwendung von Javascript-Bibliotheken, die eher in Hinsicht auf die Verwendung von vollwertigen Webservern wie *Apache* abzielen, können einige Probleme entstehen. Traditionelle Web-Ansätze verwenden häufig mehrere Sockets bei der Kommunikation mit dem Server. Wie bei allen Ansätzen bietet es sich auch bei diesem Ansatz an, die Anwendung strukturiert auf mehrere Module beziehungsweise Dateien zu verteilen. Ajax-Anwendungen werden selten als eine einzige Datei ausgeliefert. Ist die Oberfläche aus mehreren Dokumenten unter verschiedenen URLs zusammengesetzt, entscheidet der Browser, wie viele Anfragen er parallel öffnet. Durch die asynchrone Datenübertragung bei Ajax kommt es schnell zu unnötigem Polling, was eine zusätzliche Last bedeutet. Die notwendige XML-Kodierung für das X in Ajax ist nicht grundsätzlich notwendig, allerdings bei der Verwendung von Javascript-Bibliotheken oftmals unumgänglich.
- Flash (Flex/Air)
  Flash verlangt dem Server schon deutlich weniger ab. Flash-Filme sind Containerformate. Es muß nur eine einzige Datei übertragen werden. Durch die Verwendung eines eigenen Sockets inklusive eigenem Protokoll ist kein Polling auf den Server notwendig. Allerdings muß der Server die Daten an die Oberfläche XML-kodiert ausliefern. Die Verarbeitung von XML erfordert verhältnismäßig viel Speicher.
- Java (Applet/Standalone) Java-Applets werden in einem Container, dem Jar-File, ausgeliefert. Es muß nur eine einzige Datei übertragen werden. Java-Applets verwenden zur Verbindung mit dem Server ihren eigenen Socket. Auch hier ist kein unnötiges Polling notwendig. Bei Java ist die Verwendung des Sockets darüber hinaus nicht an spezielle XML-Kommunikation gebunden, es kann ein mikroprozessorfreundliches Protokoll entwickelt werden.

Wie hoch ist die Komplexität des Ansatzes?

- Ajax (HTML/Javascript)
  Ajax ist ein heterogener Ansatz aus verschiedenen Webtechnologien. Beim Ajax-Ansatz werden Formatierungsattribute für die Beschreibung von Webseiten häufig in CSS (Cascading Style Sheets) Dateien ausge-

lagert. Die ursprünglich zur Formatierung von Textinformationen verwendetete Markup-Sprache HTML wird inzwischen häufig nur noch zur Bereitstellung von Daten in einem Baum (Document Objekt Modell) verwendet. Die eigentliche Formatierung der Elemente wird durch die Verknüpfung der HTML-Daten mit den CSS-Daten mittels spezieller Metainformationen im Markup verknüpft. Javascript kann in eigenen Dateien ausgeliefert werden und hat über das Dokumenten Objekt Modell vollen Zugriff auf die Elemente des HTML-Baums. Dynamische Webseiten können bei Ajax-Lösungen vollständig durch Javascript-Code aufgebaut werden. Dies geht so weit, dass lediglich ein leerer HTML-Rumpf ausgeliefert wird. Die dynamisch veränderbaren Daten der Seite werden nach Laden des Javascriptcodes durch das Anlegen oder Verändern von Knoten im Dokumenten Objekt Modell generiert. Insgesamt entsteht eine erhöhte Komplexität durch die Vermischung unterschiedlicher Techniken, Javascript-Bibliotheken und letztendlich die Fehlerbehandlung für unterschiedliche Browser.

- Flash (Flex/Air)
  Flash liefert einen homogeneneren Ansatz. Die Verwendung von Flex zum Erstellen von Oberflächen bringt hier die Markup-Sprache MXML mit ins Spiel. Beim gleichzeitigen Einsatz von Flash und Flex müssen Symbole aus einer Flex-Datei, die in Flash genutzt werden sollen, verknüpft werden. Diese Verknüpfung wird in der Entwicklungsumgebung über einen Eigenschaften-Dialog oder im Code über spezielle Embed-Anweisungen erledigt. In Flex ist das Einbetten von Flash-Inhalten bei Verwendung von ActionScript auf gleiche und bei dem Einsatz von MXML auf ähnliche Art und Weise möglich. Das ursprünglich reine Animationswerkzeug Flash wurde im Laufe der Zeit um objektorientierte Ansätze erweitert. Eine konsequente objektorientierte Strukturierung größerer Projekte ist in Flash durch ActionScript 3 sehr einfach möglich. Intern basiert die Flash-Programmierung inzwischen nahezu vollständig auf objektorientierten Konzepten.
- Java (Applet/Standalone)
  Java liefert einen sehr homogenen Ansatz. Die Objektorientiertheit erstreckt sich über alle Bestandteile einer Anwendung. Dies betrifft die Darstellung und Formatierung der Oberfläche, wie auch die Programmlogik. Java liefert dem Programmierer von Anfang an einen klassischen objektorientierten Ansatz. Die Standardbibliothek von Java wird bereits durch das Browserplugin an den Client ausgeliefert und muß nicht durch den Server übertragen werden. Die Möglichkeiten der Standardbibliothek sind sehr umfangreich. Hierdurch ist eine Erweiterung durch optionale Bibliotheken viel später notwendig, als beispielsweise bei einem Ajax-Ansatz.

Wie einfach ist der Ansatz erweiterbar?

Die einfache Erweiterbarkeit des Ansatzes hängt bei allen Ansätzen davon ab, wie weit diese Erweiterungen zu Beginn der Entwicklung der Oberfläche eingeplant waren. Es sind die unerwarteten Erweiterungen, die die Erweiterbarkeit der Ansätze auf die Probe stellen. Wenn die Bibliothek später durch einen Benutzer erweitert wird, hängt die „Einfachheit“ der Erweiterung natürlich davon ab, welche Programmiersprachen und Techniken der Benutzer beherrscht. Hat der Benutzer eine gewisse Präferenz oder bereits Erfahrung mit einer speziellen Sprache, fällt ihm diese wahrscheinlich zunächst leichter. Der Verbreitungsgrad der verschiedenen Sprachen und Ansätze in technisch orientierten Entwicklerkreisen spielt hierbei eventuell eine Rolle.

- Ajax (HTML/Javascript)
  Durch die den Umgang mit dem Informationsmedium Internet sind viele schon einmal mit HTML in Berührung gekommen. Ungeplante Erweiterungen einer Ajax-Lösung können Anpassungen im Markup, der Logik und der Formatierung mit sich bringen. Ein erneuter ausgiebiger Test der Ajax-Anwendung zur Sicherstellung der Browserkompatiblität ist teilweise nach einer Anpassung notwendig.

- Flash (Flex/Air)
  Durch den Ursprung von Flash als reines Animationswerkzeug ist Flash eher in Bereichen des Grafikdesigns anzutreffen. Grundsätzlich ist die Erweiterung einer Flash/Flex-Anwendung nicht so kompliziert wie bei einer Ajax-Anwendung, da die Flash-Programmierung intern größtenteils auf objektorientierten Konzepten basiert. Ein erneuter Test zur Browserkompatiblität ist nicht notwendig.

- Java (Applet/Standalone)
  Java wird in der Lehre von Studiengängen der Informationstechnologie häufig als Vertreter objektorientierter Konzepte verwendet. Hierdurch ist zu vermuten, dass Java eine gewisse Verbreitung in technisch orientierten Entwicklerkreisen hat. Grundsätzlich entstehen bei der Erweiterung der Oberfläche durch die vollständige und homogene Objektorientiertheit von Java keine speziellen Probleme. Einzig die durch Java vorgegebene strenge Typisierung erlaubt keine schnellen „hacks“. Der Entwickler einer Java-Anwendung ist gezwungen, eine saubere Typisierung einzuhalten. Ein erneuter Test zur Browserkompatiblität ist auch hier nicht notwendig.

Ist ein Dateizugriff möglich?

- Ajax (HTML/Javascript) Dateien können heruntergeladen werden, dynamische Schreibzugriffe sind nicht möglich.
- Flash (Flex/Air): Flash kann auf einen durch das Flash-Plugin geschützten Bereich zugreifen.
- Java (Applet/Standalone): Java kann dynamisch auf das Dateisystem zugreifen, wenn das Applet digital signiert ist.

Wird ein Browserplugin benötigt?

- Ajax (HTML/Javascript) Nein.
- Flash (Flex/Air): Ja.
- Java (Applet/Standalone): Ja.

Ist zum Erstellen (und Erweitern) der Software eine Lizenz nötig?

- Ajax (HTML/Javascript) Nein. Es gibt kommerzielle Produkte wie beispielsweise Dreamweaver, die bei der Erstellung von Webseiten behilflich sein können. Grundsätzlich sind diese bei der Entwicklung von Ajax-Anwendungen aber nicht notwendig. Browser-Plugins wie zum Beispiel „Firebug“, die beim Debuggen einer Ajax-Anwendung sehr helfen können, sind kostenlos erhältlich.
- Flash (Flex/Air): Eine Lizenz von Adobe Flash, Adobe Flex oder beidem ist bei der Entwicklung von Flash-Anwendungen sehr dienlich. Es gibt zwar inzwischen das freie Flex-SDK und den Opensource-Actionscriptcompiler „MTASC“ [12], jedoch ist der Weg einer Flash-Entwicklung ohne eine kommerzielle Entwicklungsumgebung ein sehr steiniger.
- Java (Applet/Standalone): Mit Netbeans und Eclipse ist die Konkurrenz an kommerziellen Werkzeugen zur Java-Entwicklung nahezu vollständig vom Markt verdrängt worden. Die beiden Werkzeuge bieten alles, was zur Erstellung von Java-Applets notwendig ist. Netbeans bietet sogar einen integrierten GUI-Editor.

[12] http://www.mtasc.org/

Bewertung

	Gewichtung	Java	Flash	Ajax
Prozessorlast	5	5	3	1
Komplexität	3	3	3	1
Erweiterbarkeit	4	4	4	1
Browserpluging notwendig	1	1	1	5
Softwarelizenzkosten	2	5	1	5
Summe (höher=besser)		**61**	**43**	**27**

Bei der Verwendung von Javascript entstehen viele Probleme. Ein großes Problem ist das immer noch andauernde Kompatiblitätsproblem der unterschiedlichen Browser. Zwar hat sich hier im Laufe der Zeit sehr viel getan, jedoch gibt es immer noch einige Inkompatiblitäten zwischen den verschiedenen Browsern, die einen gehörigen Entwicklungsaufwand mit sich bringen. Eine Ajax-basierte Anwendung muß rigoros getestet werden, um so die Eigenarten der diversen Webbrowser entsprechend behandeln zu können. Aus diesem Grunde ist die Verwendung einer Bibliothek wie ExtJs unumgänglich. Eine ExtJs-Anwendung ist allerdings alles andere als ressourcenfreundlich. Im Hintergrund laufen einige aktive Warteschleifen, die notwendig sind um gewisse Effekte und Mechanismen realisieren zu können, die sonst nicht möglich wären. Eine solche Anwendung verlangt einem Browser sehr schnell 200 MB Arbeitsspeicher ab. ExtJs allein jedoch reicht nicht aus, um Graphen zu zeichnen. Hierzu müsste also eine Bibliothek wie Dojo zum Einsatz kommen. Das Mischen von verschiedenen Javascript-Bibliotheken ist allerdings auch ein Glücksspiel. Es drohen Seiteneffekte und Inkompatiblitäten, da die meisten großen Bibliotheken lieber „allein“ im Browser sind. Flash ist bei der Darstellung von grafischen Anwendungskomponenten auf jeden Fall der schnellste Kandidat. Flash ist seit je her darauf optimiert worden, beschleunigte Grafikdarstellungen zu liefern. Für Flash wäre allerdings mindestens eine Entwicklungsumgebung wie Adobe Flash™ notwendig, wenn nicht auch noch der Adobe Flex Builder™. Die Flash-Programmierung mit Actionscript ist weitestgehend auf ereignisgesteuerte Programmierung ausgelegt. Es gibt einige Dinge wie zum Beispiel Timing, die in Flash aufgrund seiner Geschichte grundsätzlich anders gelöst sind. Hier kann Flash seine Herkunft als Animationswerkzeug nicht leugnen. Die Notwendigkeit der Verarbeitung von XML ist ein weiteres Kriterium, das die Flash-Lösung unattraktiver werden läßt.

Java ist aufgrund seiner umfangreichen Klassenbibliothek, der Verfügbarkeit guter Oberflächenkonzepte und dem ausgereiften objektorientierten Ansatz die beste Lösung. Von allen gezeigten möglichen Ansätzen ist der Java-Ansatz sehr homogen. Eine spätere Erweiterbarkeit der Oberfläche um eigene Kontrollkomponenten und Darstellungsklassen ist am besten mit Java sicherzustellen. Auch die Möglichkeit der Entwicklkung eines mikrocontrollerfreundlichen Protokolls ist bei Java am ehesten gegeben.

5.2 Das Datenmodell

Aus der Anforderungsanalyse wissen wir schon einige essenzielle Komponenten und Mechanismen, die unbedingt Bestandteil der Bibliothek sein müssen. In der Anforderungsanalyse werden unter anderem folgende Forderungen erhoben:

- **R1.1** - Die Anwendung muß über Ereignisse wie das Verbinden, beziehungsweise Trennen der Oberfläche benachrichtigt werden. Es wird ein **Eventsystem** benötigt.
- **R1.2** - Das Eventsystem muß mindestens folgende Ereignisse signalisieren: Benutzer verbunden, Benutzer getrennt;
- **R2.6** - Das Eventsystem muß Ereignisse der Bedienelemente der Oberfläche verarbeiten.
- **R4.1** - Es ist ein Mechanismus zur Übergabe von Messdaten an die Bibliothek zu notwendig (PUSH-Modus).
- **R4.2** - Es ist ein Mechanismus zur Bereitstellung von Messdaten notwendig, mit dem die Bibliothek Messdaten abholen kann (PULL-MODUS).
- **R4.3** - Es ist ein Mechanismus für die Festlegung einer Frequenz zur Erfassung von Messdaten nach **R4.2** notwendig.
- **R4.4** - Die Bibliothek muß einen Datentypen für Messdaten bereitstellen, der die Bezeichnung und die Einheit in menschenlesbarer Form enthält.

Fügt man diese Forderungen zusammen, ergibt sich folgendes Gesamtbild:

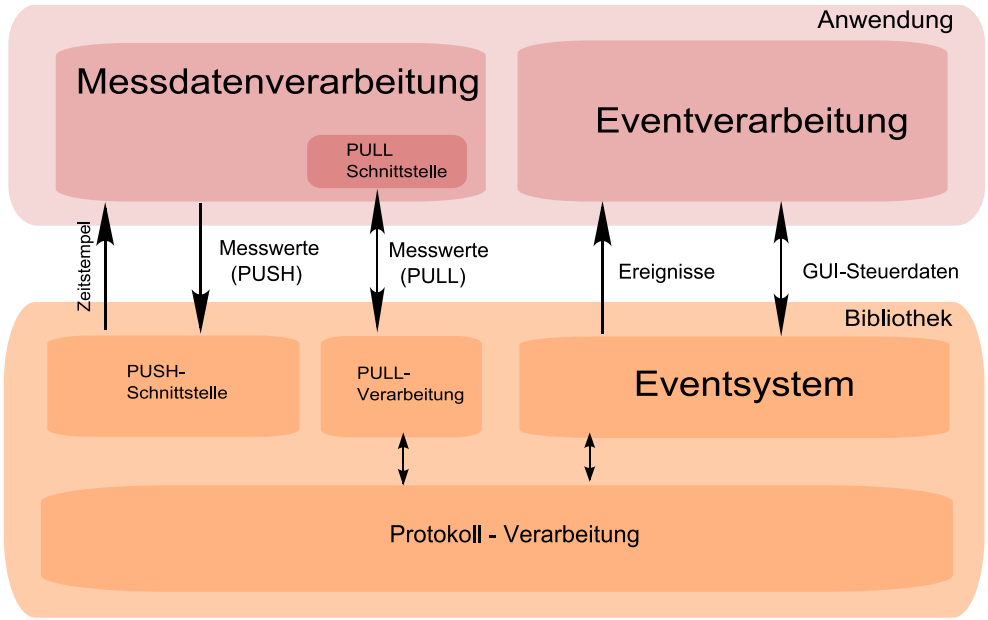


Abbildung 5.7: Datenaustausch zwischen Anwendung und Bibliothek

Abbildung 5.7 zeigt die aus der Anforderungsanalyse hervorgehenden notwendigen Komponenten und Mechanismen.

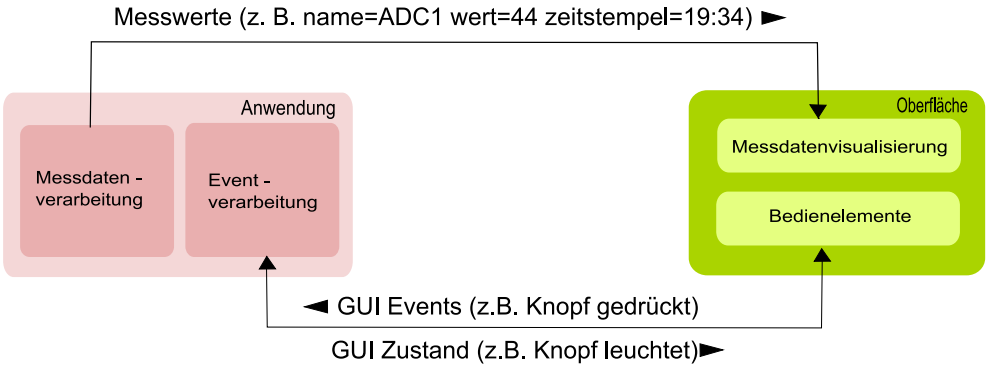


Abbildung 5.8: Datenaustausch zwischen Mikroprozessor und Oberfläche

Abbildung 5.8 zeigt die grundsätzlich zu erwartetenden Daten des Protokolls der Anwendung. In Vorraussicht auf das Protokoll sind Datentypen zu entwerfen, welche eine Basis für die Kommunikation zwischen der Oberfläche und dem Mikrocontroller bilden. Um einen geordneten Zugriff auf die Bibliothek und eine Weiterverarbeitung in Protokoll und Oberfläche zu ermöglichen ist es sinnvoll, eine Liste einzuführen, in der alle Messquellen zusammen mit ihren Attributen abgespeichert werden. So können etwaige Bibliotheksaufrufe schnell mit einer registrierten Quelle assoziiert werden. Ebenso sind Listen der vereinbarten Bedienkontexte und deren Bedienelementen notwendig, damit sowohl die Anwendung, als auch die Oberfläche an zentraler Stelle aus einem gemeinsamen Datenpool initialisiert werden können. Die Oberfläche muss erfahren, welche Messquellen es gibt. Außerdem muss sie erfahren, welche Bedienkomponenten es gibt und welchem Bedienkontext sie zuzuordnen sind.

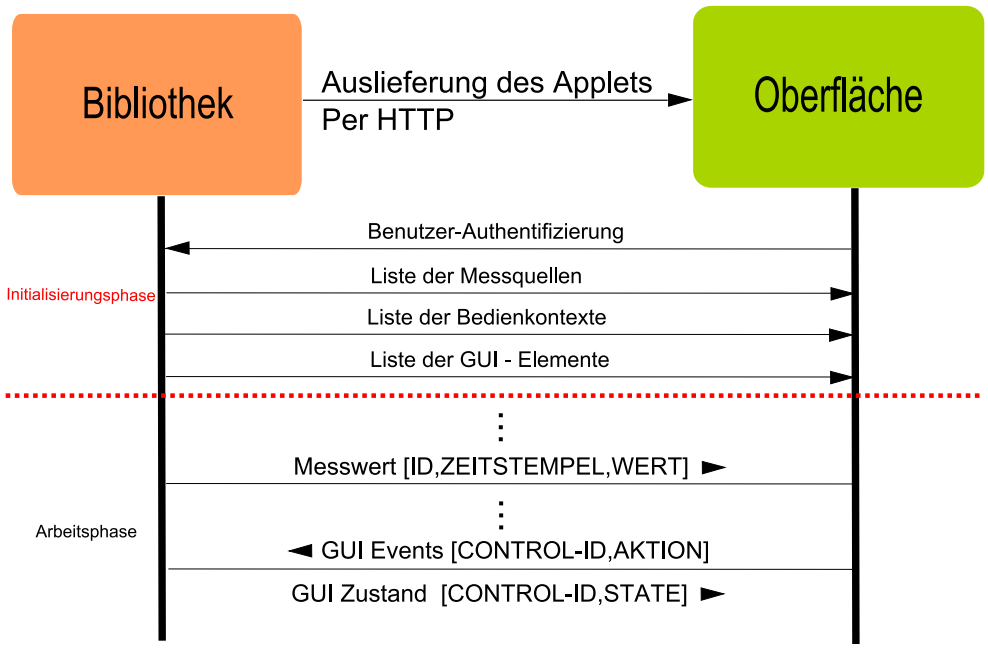


Abbildung 5.9: Ansatz zur Initialisierung der Oberfläche

Abbildung 5.9 zeigt einen Ansatz, in dem die Oberfläche bei Verbindungsaufnahme während einer Initialisierungsphase die notwendigen Daten übermittelt bekommt. Ein weiterer möglicher Ansatz wäre, die Daten gleich zusammen mit dem Applet zu liefern. Applets werden oft als JAR-Archiv ausgeliefert, was nichts anderes ist, als ein komprimiertes ZIP-Archiv. Es wäre also möglich, dass der Entwickler, wenn er sein Datenmodell fertiggestellt hat, die Beschreibiung des Modells gleich zu dem JAR hinzufügt. Der einzige Vorteil hierbei wäre allerdings, dass die Daten komprimiert übertragen werden. Da das System über eine Benutzerauthentifizierung verfügen soll ist es außerdem nicht ratsam, grundsätzlich allen Benutzern die Beschreibung der verfügbaren Messquellen und Bedienkontexte auszuliefern, da auf diesem Wege ein nicht authentifizierter Angreifer sehr schnell Informationen über das System geliefert bekommt, wenn er das Applet lädt. Selbstverständlich entsteht Sicherheit nicht allein durch die Geheimhaltung der verfügbaren Steuerkomponenten. Sicherheit entsteht nur, wenn der Benutzer gern wissen darf, was es alles an Komponenten (und somit Steuerbefehlen) gibt, aber dennoch nichts damit anfangen kann, da das System diese Befehle einfach ignoriert wenn der Benutzer sich nicht ausreichend authentifiziert. Das System sollte dennoch keine Informationen zu Messquellen oder Bedienelementen höherer Zugriffsebenen ausliefern, oder etwa Steuerbefehle dieser Komponenten entgegennehmen, wenn die Authentifizierung des Benutzers nicht ausreicht. Durch die alleinige Übertragung der für die Bedienkomponenten des Benutzers notwendigen Beschreibungsdaten, gibt das System nur die notwendigsten Informationen preis. Im Folgenden wird ein Datenmodell vorgestellt, welches sowohl notwendige Elemente für die Entwicklung eines Protokolls, als auch die bei der Verwendung einer Bibliotheksschnittstelle notwendigen Daten enthält. Dieses Datenmodell enthält Bestandteile, welche in späteren Kapiteln genauer erläutert werden.

5.2.1 Messquellen

Das Datenformat für Messquellen muss die zugrundeliegenden Daten für den Entwurf eines Protokolls und für die Verarbeitung und Eingliederung von Messquellen in das Benutzersystem bereitstellen. Es gibt Push- und Pull-Quellen. Pull-Quellen werden mit einer Pull-Periode automatisch von der Bibliothek abgeholt. Messquellen werden von der Oberfläche in einer Oszilloskopartigen Umgebung dargestellt. Jede Messquelle hat ihren eigenen Bereich. Um die Oberfläche um weitere Darstellungsweisen erweitern zu können, ist eine Datentypvereinbarung notwendig. Die Datentypvereinbarung wird in der Oberfläche mit einer konkreten Darstellungsart assoziiert. Durch die Anforderungsanalyse ist bekannt dass Datentypen parametrisierbar sein müssen. Beispielsweise muß bei normalen Funktionsplots laut der Anforderungsanalyse eine Wertebereichs-Skalierung möglich sein. Dies betrifft aber nur Funktionsplots. Selbst hinzugefügte Darstellungsarten könnten andere Anforderungen an die Parametrisierbarkeit stellen. Alle parametrisierbaren Aspekte in eigenen Datenfeldern abzuspeichern ist nicht sinnvoll, da jederzeit neue hinzukommen könnten. Aus diesem Grunde gibt es das Eigenschaften-Feld. In diesem Feld werden alle parametrisierbaren Aspekte einer Darstellungskomponente abgespeichert.

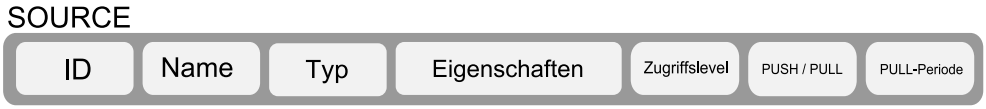


Abbildung 5.10: Das abstrakte Datenmodell einer Messquelle

- **ID** - Für die einfache Verarbeitung in Protokoll , Bibliotheksschnittstelle und Oberfläche ist eine ID notwendig.
- **Name** - Zur Darstellung auf der Oberfläche
- **Typ** - Für die Auswahl einer geeigneten Darstellungsweise auf der Oberfläche
- **Eigenschaften** - Das Eigenschaften Feld ist eine Map mit Key-Value assoziation. Damit der Umgang mit den Eigenschaften (die ja irgendwo im Sourcecode der Anwendung vereinbart werden) gut von der Hand geht, wird ein String-Format gewählt. Der reguläre Ausdruck für ein Element lautet: `[\w]+='[^']+'` Ein etwaiger String aus zwei Elementen könnte so aussehen: „`key='val' param1='120'`“. Dieser wird auf Java-Seite durch einen Parser in ein java.util.Properties-Objekt umgewandelt und der Darstellungsklasse beim Erzeugen übergeben.
- **Zugriffslevel** - Alle Messquellen sollten mit einem Zugriffslevel versehen werden können.
- **PUSH / PULL** - Legt fest ob diese Messquelle im PUSH- oder PULL-Verfahren arbeitet.
- **PULL-Periode** - Legt fest in welchem Intervall eine PULL-Quelle gelesen werden soll.

5.2.2 Bedienelemente

Laut der Anforderungsanalyse muss es möglich sein, das System um anwendungsbezogene Anzeigeinstrumente zu erweitern. Als Beispiel werden Thermometer und Drehzahlmesser genannt. Diese Komponenten gehören zu den Bedienelementen, da sie zusammen mit Knöpfen und Reglern im Bedienkontext angezeigt werden. Bedienkomponenten in der Oberfläche werden mit eigenen Nachrichten versorgt(siehe CONTROL-STATE-Nachricht in Abschnitt 5.4). Die angezeigten Daten werden jedoch höchstwahrscheinlich auch als Messquelle im System registriert sein und somit bereits an die Oberfläche gesendet werden. Die Sensitivitätsliste dient dem einfachen Zweck, ein Observer-Pattern für Bedienelemente in der Oberfläche zu realisieren. Bei der Registrierung eines Bedienelements in der Oberfläche kann das Bedienelement in ein Benachrichtigungssystem aufgenommen werden, welches eine Benachrichtigung auslöst, wenn Daten bei einer assoziierten Messquelle eintreffen. Die Eigenschaften dienen analog zu den Eigenschaften von Messquellen der Parametrisierung des im Feld Typ assoziierten Komponententyps.

Abbildung 5.11: Das abstrakte Datenmodell eines Bedienelements

- **ID** - Für die einfache Verarbeitung von Bedienelementen in Bibliotheksschnittstelle, Protokoll und Oberfläche ist eine ID notwendig.
- **Kontext ID** - Darstellungskomponenten müssen einem Kontext angehören. Dieses Feld beschreibt, welchem Bedienkontext diese Komponente angehört.
- **Name** - Zur Darstellung auf der Oberfläche
- **Typ** - Für die Auswahl eines Komponententyps (Knopf, Schieberegler, Eigene Komponente)
- **Sensitivitätsliste** - Die Sensitivitätsliste enthält IDs von assoziierten Messquellen.
- **Eigenschaften** - Bedienelemente müssen parametrisierbar sein. Zum Beispiel kann ein Thermometer mehrere Temperaturbereiche definieren, die mit unterschiedlichen Farben hinterlegt sind. Aus diesem Grunde gibt es das Eigenschaften-Feld. In diesem Feld werden alle parametrisierbaren Aspekte eines Bedienelements abgespeichert. Dieses Feld ist auf die gleiche Weise formatiert wie das Eigenschaften-Feld von Messquellen.

5.2.3 Bedienkontexte

Bedienkontexte fassen Bedienelemente zusammen und integrieren diese in das Zugangsberechtigungssystem. Jedes Bedienelement muss einem Bedienkontext zugeordnet sein. Ein Bedienkontext kann einen Anwendungsfall darstellen, wie zum Beispiel „Konfiguration“, „Wartung“ oder „Betrieb“. Jeder Kontext ist über einen hierarchischen Zugriffslevel abgesichert. In einem Benutzersystem könnten Benutzer für den Entwickler, authorisiertes Servicepersonal und den Betreiber des Systems eingerichtet werden. Authentifizierte Benutzer dürfen Bedienkontexte ihres Zugriffslevels und niedriger priorisierter Zugriffslevel bedienen. Für die Auswahl einer geeigneten Darstellungsweise eines Bedienkontextes dient das Typ-Feld. Über dieses Feld kann in der Oberfläche ein geeigneter Rahmen für die Bedienelemente festgelegt werden. Bedienkontexte können zum Beispiel Popup-Fenster sein, typischerweise sollten sie aber eher als Tab im (im Oberflächenkonzept in Kapitel 5.1 vorgestellten) Darstellungsbereich für Bedienkontexte untergebracht werden. Die Anordnung von Bedienelementen wird innerhalb eines Bedienkontextes festgelegt. Für die automatische Anordnung von Bedienelementen gibt es in vielen Oberflächenbibliotheken geeignete Layoutmanager. Es ist vorgesehen, dass über die Datenfelder im Eigenschaften-Feld ein geeigneter Layoutmanager und dessen Formatierungsanweisungen festgelegt werden können. Dieses Feld wird ebenfalls mit dem im Messquellen-Datentypen festgelegten Datenformat beschrieben.

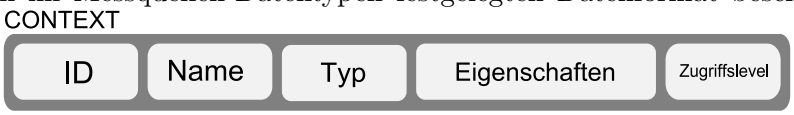


Abbildung 5.12: Das abstrakte Datenmodell eines Bedienkontexts

- **ID** - Für die einfache Verarbeitung von Bedienkontexten in Protokoll und Oberfläche ist eine ID notwendig.
- **Name** - Zur Darstellung auf der Oberfläche
- **Typ** - Für die Auswahl einer geeigneten Darstellungsweise auf der Oberfläche, beispielsweise als Tab oder Popup-Fenster
- **Eigenschaften** - Bedienkontexte fassen Bedienelemente zusammen. Für die Darstellung des Kontexts können hier Formatierungsanweisungen abgelegt werden. Dieses Feld ist auf die gleiche Weise formatiert wie das Eigenschaften-Feld von Messquellen.
- **Zugriffslevel** - Da in der Anforderungsanalyse ein hierarchisches Zugangsberechtigungssystem gefordert ist, sollten alle die Bedienkontexte mit mit einem eigenen Zugriffslevel versehen werden können.

5.2.4 Verknüpfung der Daten des Modells

Da die drei sobeben vorgestellten Datentypen die Basis für die gesamte Kommunikation und Koordination der Bibliothek bilden, ist es sehr wichtig, dass bei dem Hinzufügen eines Elements in das Modell eine strenge Validierung vorgenommen wird. Ist die Konsistenz des Datenmodells vor Beginn (bei Initialisierung der Bibliothek) sichergestellt, kann sehr viel Code zur Fehlerbehandlung (auch im Protokoll) vermieden werden.

Bestandteile der Validierung müssen mindestens sein:

- Es darf keine Quellen-ID, Kontext-ID oder Bedienelementen-ID doppelt vergeben werden.
- Die Periode einer Pull-Quelle darf nicht 0 sein und muß ein ganzzahliges Vielfaches der effektiven Timerauflösung (Stempelauflösung) der Bibliothek sein.
- Die Namen aller Datentypen sollten eindeutig vergeben sein (Nur für Oberfläche. Unkritisch, jedoch wünschenswert)
- Bedienkomponenten müssen Kontexten zugeordnet sein. Es muß einen Kontext mit der ID des Kontext-ID Felds geben.
- Die IDs in den Sensitivitätslisten der Bedienelemente sollten in der Quellenliste auch vorhanden sein(Unkritisch, jedoch wünschenswert)
- Die Strings in den verwendeten Datentypenbeschreibungen müssen terminiert sein.

Die gesamte Datenverarbeitung auf beiden beteiligten Seiten der Netzwerkkommunikation basiert auf den in diesem Modell beschriebenen Daten. Die Mikrocontrollerseite muß die Bibliothek beim Start der Anwendung mit diesen Daten initialisieren. Die hierfür notwendigen Datenstrukturen könnten zur Laufzeit oder zur Compilezeit mit Elementen gefüllt werden. Werden die Datenstrukturen im Arbeitsspeicher abgelegt, wäre es beispielsweise möglich, im Laufe der Anwendung – also nach der Initialisierungsphase – bei Bedarf weitere Datenquellen und eventuell sogar Bedienelemente hinzuzufügen. Die Speicherung aller Tabellen im Arbeitsspeicher kann allerdings verhältnismäßig teuer werden. Der Arbeitsspeicher der verwendeten Plattform hat einen Flash-Speicher von 512 Kilobyte und einen Arbeitsspeicher von 64 Kilobyte. In den bisher spezifizierten Anwendungsfällen bestand kein Bedarf, zur Laufzeit weitere Komponenten hinzuzufügen. Die Speicherung im Flash-Speicher spart dagegen wertvollen Arbeitsspeicher.

Im Kapitel 5.4.1 werden die Nachrichten zur Übertragung des Datenmodells (Auch Katalog genannt) beschrieben. Die in diesem Kapitel beschriebenen Datentypen sollen zunächst als abstrakte Beschreibung zum besseren Verständnis des Modells dienen.

5.3 Mikroprozessor-Software

Im Folgenden soll zunächst ein Bild darüber entstehen, welche Komponenten auf einem eventuellen Zielsystem zusammen kommen. Bei einer Mess- und Regelanwendung sind mindestens Sensoren und Aktoren beteiligt. Für Anwendungen, die genaue zeitliche Spezifikationen einhalten müssen, sind meist auch Timer in Verwendung. Selbstverständlich wäre es möglich, eine Timer-API bereitzustellen, eine Regler-API bereitzustellen oder eine Mess-API bereitzustellen, welche die funktionalen Komponenten des Systems zunächst in die Bibliothek integriert, um diese dann der Anwendung zur Verfügung zu stellen. Dieser Ansatz in der Art einer Laufzeitumgebung würde Anwendungen in gewisser Weise plattformunabhängig machen und hätte den Vorteil, einige Aspekte bei der Verwendung der Bibliothek transparent zu machen. Der Entwickler wäre allerdings zunächst genötigt, alle seine verwendeten Komponenten über eine Komponenten-API bei der Bibliothek zu registrieren. Dieser Ansatz widerspricht den Anforderungen an die Bibliothek. Die resultierende Bibliothek soll auch nachträglich in bestehende Systeme integrierbar sein und der Anwendung so wenig Vorschriften machen, wie notwendig.

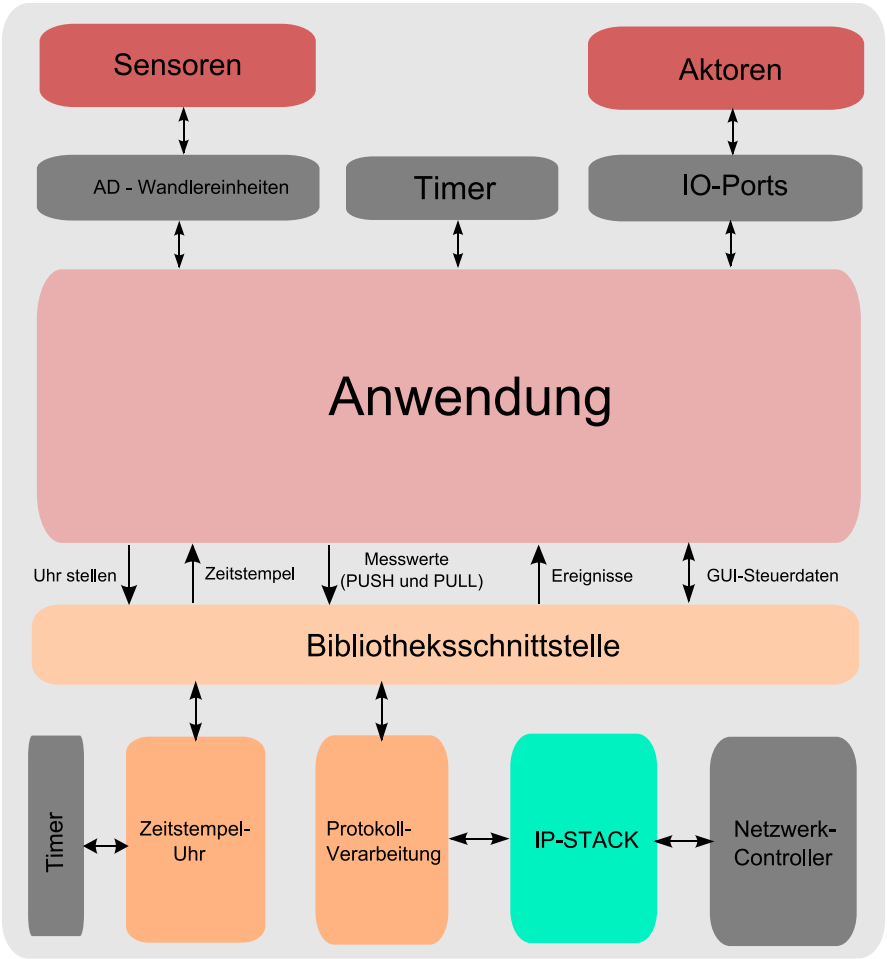


Abbildung 5.13: Soft-und Hardwarekomponenten der Mikrocontroller-Seite

Abbildung 5.13 zeigt eine grobe Übersicht über die beteiligten Soft- und Hardwarekomponenten auf einem Mikrocontroller. Die Sensoren, Timer und Aktoren werden allein von der Anwendung verwaltet.

5.3.1 Zeitstempel-Uhr

Damit absolute und relative Zeiten im System angegeben und gemessen werden können ist eine Zeitstempel-Uhr notwendig. Als de dacto Standard für Zeitstempel gilt die Unix-Zeit. Sie wurde 1969 für das Betriebssystem Unix entwickelt und als die Anzahl der vergangenen Sekunden seit dem 1.Januar 1970 definiert. Dieses Datum wird auch als *Epoch* bezeichnet. Es ist natürlich zunächst verwunderlich, dass ausgerechnet 1970 der Nullpunkt ist. Dies hat aber mehrere Gründe. Bei der Speicherung von Zeitstempeln als Integer kam zur Zeit der Einführung vor allem das 32-Bit-Format in Frage. mit dem Wertebereich von 0 - 2^{32} (4 294 967 296) läßt sich bei einer Speicherung von Sekunden eine Zeitspanne von ca 136 Jahren und vorzeichenbehaftet eine Zeitspanne von 68 Jahren darstellen. Hätte man beispielsweise das Jahr 0 (Christi Geburt) genommen, wäre der Wertebereich von 32Bit nicht ausreichend gewesen, um die Gegenwart zu erfassen. Dies gelingt nur mit Sekunden seit 1970, was übrigens nur noch bis zum Jahr 2106 funktioniert. Inzwischen werden auch häufig die Zeitstempelformate Millisekunden oder Mikrosekunden seit dem 1. Januar 1970 verwendet. Die maximale Zeitstempelauflösung beträgt laut der Anforderungsanalyse $1\mu s$, damit die Bibliothek auch für zukünftige Ansprüche skalierbar bleibt. Die Zeitstempelauflösung muss aber nicht voll ausgeschöpft werden. Eine Zeitstempel-Uhr kann mit einer langsameren Periode ticken und jeweils die Periodendauer auf die Uhr heraufzählen. Damit die Interruptlast eines Systems nicht zu hoch wird, muß abgewägt werden, welche Zeitstempelauflösung auf der Plattform überhaupt realistisch ist. Als Beispiel die Eckdaten der verwendeten Plattform:

ARM7TDMI-S (Dreistufige Pipeline)
Codeausführung $\approx$ 1,9 Takte pro Intstruktion.(Siehe[26, s.5])
Frequenz F = 72 MHz

Dauer eines Clock-Zyklus t = $\frac{1}{F} = \frac{1}{72*10^6 Hz} = 13,88ns$
Dauer einer AD-Wandlung auf dem LPC2468: $2,5\mu s$

$1\mu s \hat{=}$ 72 Cpu-Takte $\approx$ 37 Instruktionen
$10\mu s \hat{=}$ 720 Cpu-Takte$\approx$ 378 Instruktionen
$100\mu s \hat{=}$ 7200 Cpu-Takte$\approx$ 3789 Instruktionen

Maximale Abtastrate bei $100\mu s \doteq 10KHz$
Kleinste erkennbare Signalfrequenz: $\leq 5KHz$ (nach Shannon)

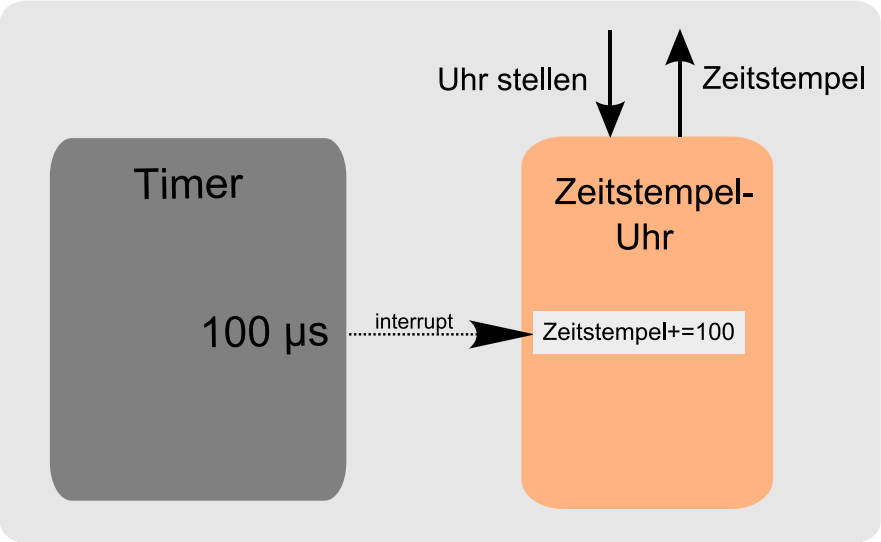


Abbildung 5.14: Die Zeitstempel-Uhr

Ein Hardware-Timer verfügt normalerweise über ein Zählregister, dass mit einem externen Takt hochgezählt wird. Bei dem verwendeten Controller ist das Register zwar immerhin 32 Bit breit, jedoch reichen 32 Bit nicht aus, um absolute Zeit in Mikrosekunden seit 1970 darzustellen. Normalerweise bedient man sich in solch einem Falle einer Zählvariable, die durch das Auftreten eines Timer-Interrupts hochgezählt wird. Der Timer wird auf eine angenehme Dauer wie zum Beispiel $100\mu s$ eingestellt. Für die Breite der Zählvariable bieten sich insbesondere Bitbreiten von 2^{8*n} an, da Rechenoperationen auf diesen Größen ohne weitere Vorverarbeitung durch die CPU ausgeführt werden können. Ist die verwendete Bitbreite ein ganzzahliges Vielfaches der Bitbreite der verwendeten CPU, lassen sich Rechenoperationen sehr einfach über mehrere Register verteilen. Die Addition erfolgt über die üblichen Assembleroperationen `ADD` und `ADD.C` (Add with Carry). Tabelle 5.3.1 zeigt die möglichen Zeitspannen für mögliche Integer-Typen.

bits	Mikrosekunden	Zeitspanne	max. Datum
2^8	256	256 Mikrosekunden	01.01.1970 00:00:00
2^{16}	65536	65 Millisekunden	01.01.1970 00:00:00
2^{32}	4294967296	ca 71 Minuten	01.01.1970 01:11:34
2^{63}	9223372036854775807	ca 290301 Jahre	10.01.294247 04:00:54

Anhand der Tabelle erkennt man, dass ein 64-Bit Zeitstempel für diese Anwendung mehr als ausreichend ist. In der Tabelle ist der 63-Bit Wert angegeben, da der korresbondierende primitive Typ `long` wie alle Java-Typen vorzeichenbehaftet ist. Sobald die Uhr gestellt wurde, kann jederzeit der aktuelle Zeitstempel ausgelesen werden. Zirka alle 71 Minuten findet ein Carry auf die obere Hälfte des Zeitstempels statt. Da die Zugriffe auf einen 64-Bit Wert auf einem 32-Bit System allerdings nicht atomar sind, ist es notwendig, eine Ausleseroutine zu implementieren, welche den Timerinterrupt für den Zugriff auf den Zeitstempel kurzzeitig deaktiviert. Damit hierbei der Timer nicht ungenau

wird, ist ein Timer zu verwenden, welcher einen Reset-on-Match Modus besitzt. Bei einem solchen Timer kann der Timer-Interrupt ausgeschaltet werden ohne das der Timer nicht mehr „aufgezogen“ würde. Der Timer-Interrupt liegt in Form eines Interrupt-Flags im Interrupt-Flag-Register der CPU an und löst aus, sobald der Interrupt wieder aktiviert wird. Der Timer-Interrupt muß innerhalb der eingestellten Timer-Periode wieder aktiviert werden, damit keine Zeit verloren geht. Bei einer Timer-Auflösung von $100\mu s$ ist dies auf jeden Fall kein Problem.

Damit die Anwendung die Uhr der Bibliothek stellen kann, benötigt die Plattform möglichst eine batteriegepufferte Uhr, damit das System nicht nach jedem Reset die Uhrzeit vergisst. Der in der Realisation verwendete Prozessor LPC2468 verfügt über eine sehr einfach zu verwendende Uhr[13].

Die Unixzeit verwendet keine Schaltsekunden. Es ist allerdings notwendig, bei der Umrechnung darauf zu achten, dass die Schaltjahre berücksichtigt werden. Laut Definition besitzen normale Jahre 365 Tage. Ein Schaltjahr besitzt 366 Tage. Ein Schaltjahr ist ein Jahr, welches durch vier, aber nicht durch hundert teilbar ist. Jahre die durch 400 teilbar sind sind wiederrum trotzdem Schaltjahre. Dies resultiert aus der Tatsache, dass ein Jahr nicht genau 365.25 Tage, sondern 365,24219052 Tage zählt. Als Beispiel wird die Fallunterscheidung für Schaltjahre in Pseudocode angeführt.

```
zeitstempel=0
Tag = 24 * 60 * 60 * 1000 * 1000


für alle Jahre bis (aktuelles Jahr-1):
if(jahr%4 == 0 & jahr%100!=0) zeitstempel+= 366 * Tag
else if(jahr%400==) zeitstempel += 366 * Tag
else zeitstempel += 365 * Tag
```

Unter Berücksichtigung des Schaltjahres kann auf ähnliche Weise mit den Monaten, Tagen, Stunden, Minuten und Sekunden des aktuellen Jahres vorgegangen werden. Um diese Berechnung zu beschleunigen, kann der Zeitstempel des Entwicklungszeitpunkts verwendet werden, um das Verfahren nur auf den seitdem vergangenen Zeitraum anwenden zu müssen. Das Stellen der Uhr über die Oberfläche kann beispielsweise durch ein Bedienelement in einem abgesicherten „setup“-Kontext hinzugefügt werden. Dieser sollte allerdings lieber die System-Uhr des Mikroprozessors stellen und dann eine Synchronisierung mit der Zeitstempeluhr mittels des oben genannten Verfahrens einleiten. Absolute Zeitstempel müssen auch einen Datenaustausch zwischen verschiedenen Zeitzonen unterstützen. Aus diesem Grunde muß die Uhrzeit auf dem Mikroprozessorsystem in UTC (Koordinierte Weltzeit), beziehungsweise GMT (

[13] *Mikrocontroller.net:* `http://www.mikrocontroller.net/articles/ARM-elf-GCC-Tutorial#Realtime_Clock_.28RTC.29`

Greenwich Mean Time) angegeben werden. Die Zeitstempel werden im Client für die Darstellung in die lokale Zeitzone umgerechnet.

5.3.2 Erfassung von Messdaten

Die Darstellung von Messdaten ist ein wesentlicher Bestandteil der Bibliothek. Die Messdaten müssen mit Zeitstempeln versehen und an die Oberfläche übertragen werden. Eine Bereitstellung eines Mechanismus zur Übergabe von Messwerten an die Bibliothek ist ebenso vorzusehen, wie ein Mechanismus zur automatischen periodischen Abholung von Messdaten durch die Bibliothek. Die beiden Modi werden im Folgenden Push-Modus und Pull-Modus genannt.

Messquellen können Messdaten, aber auch spezielle Prozessdaten ausgeben. Es gibt keinen Grund dafür, allen Messquellen ein einheitliches Datenformat aufzuzwängen. Wie aus der Anforderungsanalyse hervorgeht, sollten Messquellen grundsätzlich unterschiedliche Größen haben können. Die weitere Verarbeitung von Messwerten zur Darstellung sollte weitestgehen der Oberfläche überlassen werden, um die Ressourcen auf der Mikroprozessorplattform zu schonen. An dieser Stelle sei nur kurz auf die Erweiterbarkeit der Darstellung um neue Darstellungstypen hingewiesen (weitere Informationen hierzu folgen in Abschnitt **??**). Es sind durchaus Datentypen vorstellbar, die Daten variabler Größe vorsehen. Dieser Abschnitt befasst sich nur mit der Erfassung dieser Daten.

- Messdaten werden nicht für die Oberfläche vorverarbeitet.
- Verschiedene Messquellen können unterschiedlich lange Daten liefern.
- Eine Messquelle kann unterschiedlich lange Daten liefern.
- Die Interpretation der Daten ist Aufgabe der Darstellungstyp-Klasse in der Oberfläche

5.3.3 Push-Modus

Der Push-Modus ist sicher die einfachste Art, Messdaten an die Bibliothek zu übergeben. Die Bereitstellung einer Bibliotheksfunktion zur Übergabe eines Messwerts (im Folgenden auch Sample genannt) an die Bibliothek reicht an dieser Stelle aus. Da es sich bei den übergebenen Daten um unterschiedlich lange Daten handelt, sollte ein Pointer auf die Daten übergeben werden. Damit die Bibliothek weiß wie lang diese Daten sind, muss bei der Übergabe außerdem die Länge der Daten angegeben werden. Damit eine Zuordnung zu einer registrierten Messquelle stattfinden kann, ist die ID der Messquelle anzugeben. Die Signatur dieser Funktion könnte also in etwa so aussehen:

```
push(ID,SAMPLE*,LENGTH)
```

Die Zeitstempel für die Übertragung werden automatisch durch die Bibliothek hinzugefügt. Es sind auch Anwendungsfälle vorstellbar, bei denen der Benutzer der Bibliothek Kenntnis von dem Zeitstempel oder einen Einfluss auf diesen haben möchte. Zeitstempel könnten durch die Anwendung zum Beispiel geringfügig auf- oder abgerundet werden. Diese Anforderung ist leicht zu erfüllen. Hierzu kann zunächst ein Zeitstempel von der Zeitstempel-Uhr geholt werden. Es fehlt nur noch eine Funktion, die die Übergabe eines Messwertes mit Zeitstempel ermöglicht. Die Signatur dieser Funktion könnte in etwa so aussehen:

```
push(ID,TIMESTAMP,SAMPLE*,LENGTH)
```

5.3.4 Pull-Modus

Der Pull-Modus ermöglicht die automatische periodische Abholung von Messdaten. Die kleinste mögliche Periode einer Pull-Quelle ist die Periode der Zeitstempel-Uhr. Hierzu wird bei der Registrierung der Messquelle angegeben, dass es sich um eine Pull-Quelle handelt und in welcher Frequenz die Daten abgeholt werden sollen. Für die periodische Abholung von Messdaten ist ein Mechanismus notwendig, welcher die Zeiten für die periodischen Abholungen der Messquellen intern verwaltet. Die in Abschnitt 5.3.1 vorgestellte Zeitstempel-UHR kann diese Aufgabe übernehmen. Die Zeitstempel-Uhr wird regelmäßig durch eine Interrupt-Service-Routine aufgerufen. Um Pull-Quellen zu verarbeiten muss die Zeitstempel-Uhr die Pull-Perioden der registrierten Pull-Quellen verwalten und gegebenenfalls die Abholung von Messdaten auslösen.

Laut der Anforderungsanalyse ist ein Mechanismus bereitzustellen welcher die Abholung von Messwerten durch die Bibliothek ermöglicht. Denkbar wären Datenstrukturen, welche für den Austausch von Messwerten zwischen der Anwendung und der Bibliothek angelegt werden. Diese Datenstrukturen müssten allerdings wiederum aktiv durch die Anwendung gefüllt werden. Ein weiterer Ansatz wäre, dass die Anwendung bei der Registrierung einer Messquelle einen Pointer auf eine interne Variable preisgibt, welche durch die Bibliothek regelmäßig gelesen werden kann. Beide Ansätze sind nicht flexibel genug.

Ein vielversprechender Ansatz ist die Registrierung eines Funktionspointers. Die Anwendung kann eine Funktion implementieren, über die die angeforderten Messdaten zeitnah berechnet und ausgegeben werden können. Auch der Pull-Modus muß variable Datenlängen verarbeiten können. Die Daten könnten entweder nach dem Aufruf in einem vereinbarten Speicherbereich liegen oder durch die Routine in einen übergebenen Speicherbereich kopiert werden. Die mögliche Signatur einer Callbackroutine könnte wie folgt aussehen:

```
int pull_quelle1(BUF* data)
```

Für die Abholung von Pull-Quellen sind zwei verschiedene Ansätze möglich.

Pull-Verarbeitung in Interrupt-Service-Routine

Im ersten Ansatz ist vorgesehen, die Pull-Verarbeitung direkt durch die Interrupt-Service-Routine zu ermöglichen. Hierdurch entsteht auf dem System eine erhöhte Interrupt-Last, die nicht mit den Timing-Anforderungen einiger Anwendungen harmonieren könnte, jedoch wird durch die Erfassung von Messdaten direkt aus der Interrupt-Service-Routine heraus eine absolut periodentreue Abtastung von Messquellen möglich. Der asynchrone Zugriff auf Anwendungsdaten aus einer Interrupt-Service-Routine heraus kann inkonsistente Daten liefern, sofern die Daten durch die Anwendung nicht in einer atomaren Operation erzeugt werden. Obwohl der Benutzer durch die Implementierung genau bestimmen kann was geschehen soll, wenn die Anwendung asynchron abgefragt wird, muß der Anwender genau wissen, was er tut wenn er diesen Modus verwendet.

Pull-Verarbeitung in zur Anwendung synchronisiertem Codeabschnitt

Im zweiten Ansatz ist vorgesehen, die Messquelle in der ISR lediglich als *zu messen* zu markieren und die Messung nach Verlassen der ISR in einem zu der Anwendung synchronisierten Codeabschnitt durchzuführen. Die Pull-Verarbeitung kann (Durch den Scheduler oder eine Superloop-Architektur) wechselseitig mit der Anwendung aufgerufen werden. Hierdurch ist die Präzision des Zeitstempels der Messung nicht gefährdet, da der Zeitstempel erst erzeugt wird, wenn die Messung letztendlich durchgeführt wird. Hierfür ist die Einführung eines exklusiven Bibliotheksaufrufs notwendig. Dieser Modus ist der sicherere, jedoch könnte die Periodendauer der Messungen geringfügig korrelieren. Die zeitliche Schwankung hängt von der Zeit ab, die vergeht, bis der erwähnte synchrone Codeabschnitt ausgeführt wird. Zeitstempel haben grundsätzlich eine begrenzte Präzision. Bei einer Zeitstempel-Auflösung von 100 μs bewegt sich diese Korrelation unterhalb der Zeitlichen Auflösung.

5.3.5 Das Eventsystem

Ein Eventsystem kann grundsätzlich zwei Arten der Kommunikation verwenden. Entweder es versorgt den Bennutzer mit Ereignissen nach der Art eines Observer-Patterns, oder es stellt die eintreffenden Ereignisse in einer Queue bereit. Die Ereignisse in dieser Bibliothek werden vor allem durch die Bedienelemente der Oberfläche ausgelöst. Die Oberfläche hat nach der Verbindungsaufnahme mit dem System in der Initialisierungsphase unter anderem eine Liste mit Bedienelementen erhalten, die die Typen und IDs der Bedienelemente enthält. Unterschiedliche Bedienelemente können unterschielich lange Ereignisdaten übermitteln. Die Ereignisdaten einer Bedienkomponente, die beispielsweise einen Knopf darstellt, können in einem einzigen Byte untergebracht werden. Dagegen könnte eine Bedienkomponente die ein Eingabefeld darstellt, Strings variabler Länge an den Mikroprozessor übertragen. Neben den Ereignissen der Bedienelemente der Oberfläche sollen über das Eventsystem auch die Ereignisse „Benutzer verbunden“ und „Benutzer getrennt“ signalisiert werden. Auch Ereignisse die über Ausnahmen der Bibliothek informieren, wie zum Beispiel einen Pufferüberlauf, gehören hier her.

Event Queues

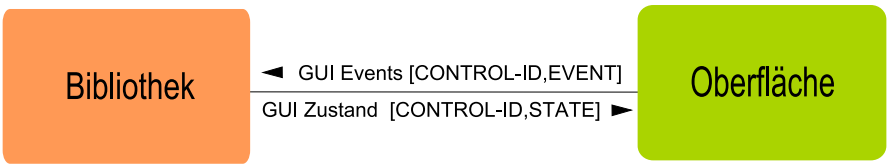


Abbildung 5.15: Eventdaten zwischen Bibliothek und Oberfläche

Die Implementierung eines Eventsystems wird selbst auf großen Systemen wie zum Beispiel bei Microsoft Windows intern auf eine Message Queue heruntergebrochen[14][15]. Programme die mit der WIN32-API arbeiten, verarbeiten sämtliche Oberflächenereignisse über einen sogenannten *Message Loop*. In diesem Message Loop werden die Events aus der Message Queue des Prozesses verarbeitet, zum Teil in komplexeren Automaten, zum Teil wird direkt auf Ereignisse reagiert. Die Programmierung von Oberflächen beinhaltet grundsätzlich immer eine zusätzliche Komplexität, die durch die Synchronisation der Anwendung mit der Oberfläche und durch die gezielte Reaktion auf Oberflächenereignisse entsteht. Um so komplexer die Oberfläche ist, desto komplexer ist die Verarbeitung aller notwendigen Ereignisse. Eine gewisse Komplexität bleibt dem Benutzer also auch in dieser Bibliothek nicht erspart. Um so komplexer eine Oberfläche ist (Je mehr Bedienelemente sie beispielsweise enthält), desto länger wird der Code, die anfallenden Ereignisse zu verarbeiten.

[14] *MSDN:*`http://msdn.microsoft.com/en-us/library/ms644927(VS.85).aspx`
[15]

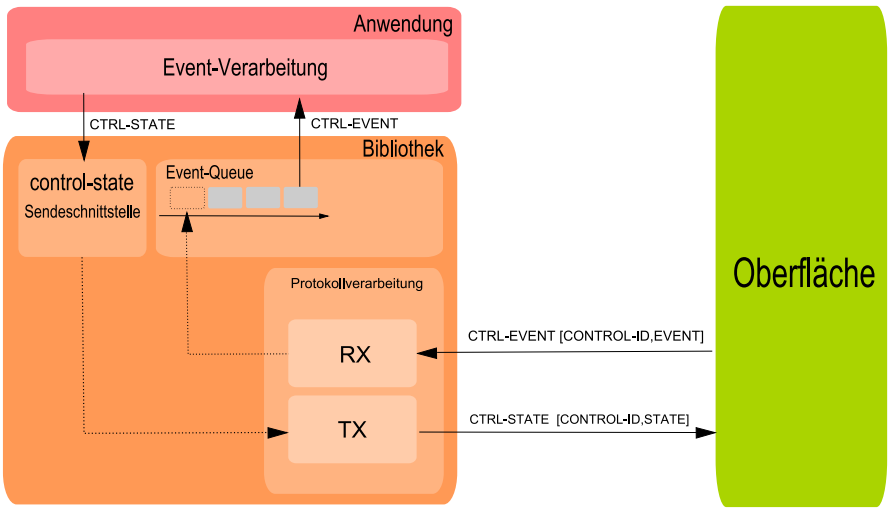


Abbildung 5.16: Eventdaten zwischen Anwendung und Bibliothek

Alle ausgewachsenen Echtzeitbetriebssysteme wie zum Beispiel FreeRTOS, QNX oder VxWorks verfügen eigene Message Queue- Standards. Die Anbindung einer Betriebssystem-Queue ist also eher dem Benutzer überlassen sofern er das wünscht. Da die Bibliothek überdies auch ohne Betriebssystem funktionieren sollte, ist die Implementierung eines eigenen Message Queue Mechanismus unumgänglich. Message Queues sind von der Grundstruktur her nicht besonders komplex. Es können hierzu Arrays von Pointern auf Datenstrukturen oder verkettete Listen von Datenstrukturen verwendet werden. Die Anwendung sollte über eine Zugriffsfunktion prüfen können, ob sich events in der Queue befinden und über eine Zugriffsfunktion das nächste Event holen. Auf diesem Wege ist es möglich, einen Message-Loop in der Anwendung zu implementieren, der terminiert, wenn die Queue leer ist. Innerhalb dieses Message-Loops kann die Anwendung simple simple Fallunterscheidungen und Automaten im Stile traditioneller Message-Loops (vgl. Microsoft) relalisieren.

Beim Zugriff auf das Ereignis muß die Anwendung die Ereignis-Daten der Bedienkomponente (CTRL-EVENT) selbst interpretieren. Die Kommunikation mit einer Bedienkomponente der Oberfläche erfolgt über dessen CTRL-STATE Nachricht und CTRL-EVENT Nachricht. Diese Nachrichten unterliegen keinen weiteren Regeln. Es sind komponentenbezogene „Subprotokolle“ denkbar, die diesen Kommunikationskanal verwenden um komplexe Mechanismen zu realisieren. Diese Daten werden Pro Bedienkomponente spezifiziert. Bei komplexeren Bedienkomponenten ist es empfehlenswert, korrespondierende Helfermodule in C zu realisieren. Die Anwendung muß nach Verbindung eines Clients über das Eventsystem eine „Benutzer-Verbunden“ - Nachricht erhalten, die über den Zugriffslevel des Clients informiert. Daraufhin kann die Bibliothek alle dem Zugriffslevel entsprechenden Oberflächenkomponenten über deren spezifische CTRL-STATE Nachricht initialisieren. Eine Priorisierung von Ereignisse in der Queue ist nicht notwendig.

CONTROL-STATE Nachrichten können jederzeit an Bedienkomponenten gesendet werden. Bedienkomponenten, die einen ausschliesslich visualisierenden Charakter haben, visualisieren meistens die wichtigsten Daten des Systems. Die wichtigsten Daten des Systems werden oftmals auch als Messquelle angemeldet sein, damit der zeitliche Verlauf dieser Parameter betrachtet werden kann. Komponenten, deren Natur es ist, Daten zu visualisieren, sollten nicht über das Eventsystem aktualisiert werden, da die Daten ohnehin schon an die Oberfläche gesendet werden.

Aus diesem Grunde gibt es in der Oberfläche ein Benachrichtigungssystem, welches Gebrauch von der Sensitivitätsliste des Bedienelements macht. Die Oberfläche kann auf diesem Wege die Bedienkomponente benachrichtigen, dass Daten einer assoziierten Messquelle angekommen sind. Um eine maximale Interoperabilität zu gewährleisten, wird Darstellungsklassen von Bedienkomponenten in der Oberfläche Zugriff auf Messquellendaten gewährt.

5.3.6 Benutzerauthentifizierung

Abbildung 5.17: Ein Benutzer Datentyp

Die Benutzerauthentifizierung basiert auf einer Tabelle welche im Flash-Speicher des Mikroprozessors abgelegt wird. Die einzelnen Datenelemente (Siehe Abbildung 5.17) sind die Basis für einen Benutzerauthentifizierung.

Der Fokus dieser Arbeit ist nicht die Netzwerksicherheit. Trotzdem soll zumindest ein gewisser Grad an Sicherheit bei der Benutzerauthentifizierung verwendet werden. Generell ist die Übertragung von Plain-Text Passwörtern über das Netzwerk nicht zu empfehlen. Da die beiden Parteien der Verbindung bereits ein Geheimnis miteinander teilen, wäre das Einfachste die Verwendung eines simplen Challenge/Response-Verfahrens(siehe [35]).

Bei einem Challenge/Response-Verfahren generiert der Server eine Zufallszahl (Challenge) und sendet diese zum Client. Der Client konkateniert diese Zufallszahl mit seinem Passwort und wendet eine kryptographischen Hash-Funktion, wie zum Beispiel MD5 SHA1 oder SHA2 auf die Daten an. Das Ergebnis sendet er zurück an den Server. Der Server prüft dieses gegen seine eigene Berechnung. Während der gesamten Datenkommunikaion wird das Passwort nicht ein einziges Mal unverschlüsselt übertragen. Einer Man-in-the-Middle Attacke hält dieses Verfahren zwar nicht stand, einer Replay-Attacke dagegen schon, da sich die Challenge-Zahl jedes Mal ändert. Die Sicherheit dieses Verfahrens ist stark von der Länge des Challenge-Werts, des Passworts, sowie der Sicherheit der verwendeten kryptographischen Hash-Funktion abhängig. Bei der Verwendung von `rand()` für Zufallszahlen, ist `srand()` zu verwenden, um einen *Seed* für den Pseudo-Zufallszahlengenerator zu setzen. Damit der Controller nicht jedes Mal die gleichen Zufallszahlen ausgibt, muss er nach jeder Ausgabe einen neuen Seed im Flash-Speicher oder auf einer SD-Karte abgelegen.

Reihenfolge der Nachrichten der Benutzerauthentifizierung

Ist das System frei, wird eine AUTHORIZATION-REQUIRED - Nachricht geschickt:

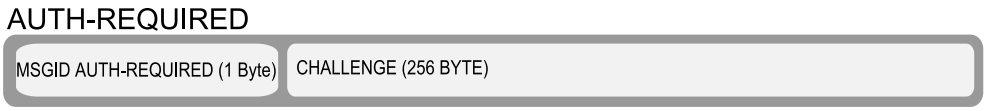


Abbildung 5.18: AUTHORIZARION-REQUIRED Nachricht

Die Oberfläche öffnet einen Dialog zur Eingabe von Benutzername und Passwort. Hat der Benutzer beides eingegeben antwortet die Oberfläche mit der AUTH Nachricht:

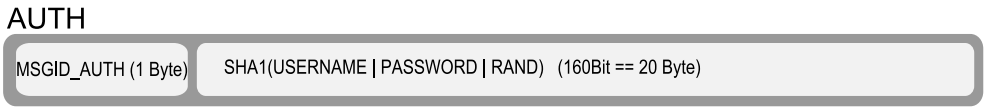


Abbildung 5.19: AUTH Nachricht

Der Server prüft die Authentifizierung. Ist die Authtentifizierung richtig, antwortet er mit:

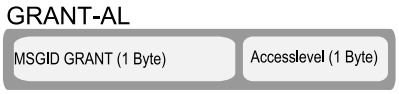


Abbildung 5.20: GRANT-AL Nachricht

Ist die Authentifizierung falsch, antwortet der Server bis zu zwei mal mit AUTHORIZARION-REQUIRED. Ist die Authentifizierung drei mal fehlgeschlagen, antwortet der Server mit ACCESS-DENIED.

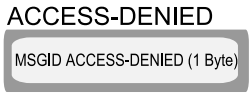


Abbildung 5.21: ACCESS-DENIED Nachricht

5.3.7 Protokollverarbeitung

Es wäre schön, wenn die Einbindung eines IP-Stacks durch den regelmäßigen Aufruf einer Stack-Funktion realisiert werden könnte. Es gibt IP-Stacks die funktionieren so und es ist nicht unlogisch, dass das so ist. Es ist die Aufgabe des IP-Stacks, eine Socket-Artige Schnittstelle zur Verfügung zu stellen, die es dem Benutzer ermöglicht Daten zu senden und zu empfangen. Viele IP-Stacks für Mikroprozessorsysteme lassen sich in Systeme integrieren, welche ohne Betriebssystem betrieben werden. IP-Stacks benötigen allerdings Rechenzeit um Ihre Aufgabe zu erfüllen. Die Ressourcen auf Mikroprozessorsystemen sind begrenzt. Durch den regelmäßigen Aufruf des IP-Stacks ist es möglich, die Rechenzeit auf dem System selbst zu verteilen.

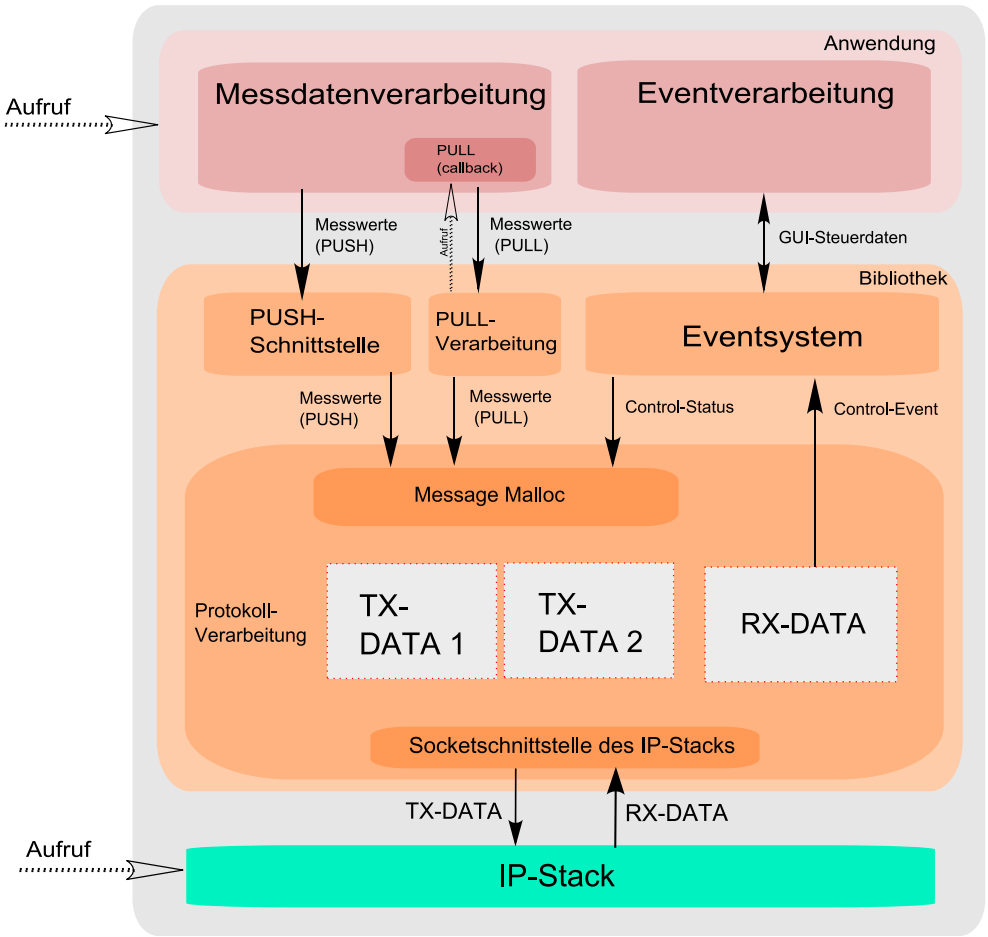


Abbildung 5.22: Einfluss von Stacks auf die Bibliotheksarchitektur

Abbildung 5.22 zeigt den Wechselseitigen Aufruf eines IP-Stacks und einer Mess- und Regelanwendung. Während eines Aufrufs der Anwendung könnte die Anwendung mehrere Messwerte (Samples) und mehrere Bedienkomponenten-Zustände an die Bibliothek übergeben. Diese Daten müssen vom Protokollverarbeitungsmodul in Nachrichen (SAMPLE-Nachrichten und CTRL-STATE Nachrichten) für die Oberfläche übersetzt und an den IP-Stack weitergegeben

werden.

Wie im Grundlagenkapitel 3.3 beschrieben, verfolgen einige IP-Stacks die Strategie, Arbeitsspeicher für Sendepuffer zu sparen indem sie dafür sorgen, dass immer nur ein Paket zur Zeit auf der Leitung ist. Bei der Verwendung solch eines Stacks ist es nicht möglich, viele kleine Pakete zu schicken, ohne die Datentransferrate dabei massiv zu reduzieren. Obwohl die Anwendung eventuell genug Daten zu versenden hätte, wird die Datenübertragung durch die kleinen Pakete ausgebremst.

Die Übertragungszeit für Oberflächennachrichten gehört nicht zu den extrem zeitkritischen Elementen der Bibliothek. Es hat keinen nennenswerten Einfluss auf die Verrichtung einer Mess- und Regelaufgabe, wenn sich beispielsweise die Zeit vom „Stempeln“ einer SAMPLE-Nachricht bis zur Auslieferung an die Oberfläche innerhalb einer gewissen Toleranz bewegt. Aus diesem Grunde werden durch das Protokollverarbeitungsmodul zunächst mehrere Nachrichten zu Nachrichpaketen zusammengefasst. Hierdurch kann die Paketgröße innerhalb einer kleinen Zeitspanne durch die eintreffenden Daten der Anwendung akkumuliert werden, um dann als größeres Paket zum Client geschickt zu werden. Sobald ein volles Ethernet-Frame von 1460 Nutzbytes (40 Bytes sind der Ethernet-Header) gefüllt ist, kann das Paket losgeschickt werden. Falls die Anwendung allerdings eine gewisse Periode keine Nachrichten erzeugt, weil keine Quellen gelesen werden und keine Oberflächen-Ereignisse stattfinden, könnten ungesendete Nachrichten längere Zeit im Protokolladapter verweilen. Damit dies nicht geschieht, wird beim Hinzufügen der ersten Nachricht ein Timer gestellt. Ist das Nachrichtenpaket vor Eintreten eines Timeouts voll, wird es an den Stack übergeben und der Timer angehalten. Nach Ablauf des Timers werden auch kleine Pakete verschickt. Dieser Ansatz geht einen Kompromiss bei der Aktualität der Oberfläche ein, um größere Datenraten bei der Übertragung von Nachrichte erreichen zu können. Der Kompromiss für die Aktualität ist allerdings nicht sehr teuer. Um ein ausgewogenes Gleichgewicht herzustellen werden maximal Timeouts von 100 ms benötigt. Ein Zeitraum von 100 ms entspricht der Dauer eines menschlichen Wimpernschlags.

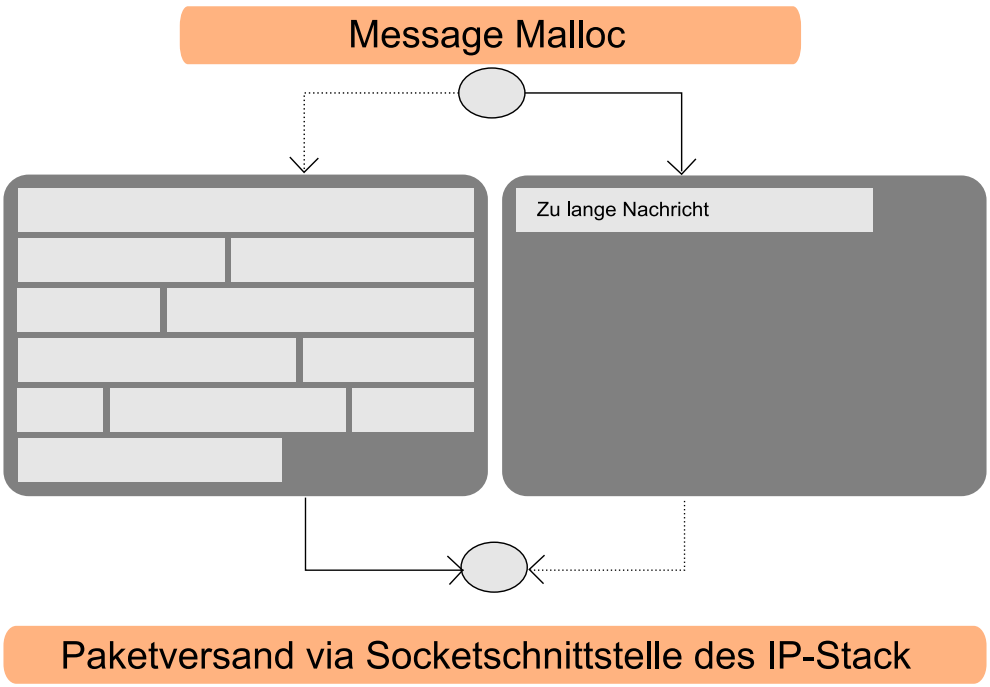


Abbildung 5.23: Pufferstrategie für Nachrichtenpakete

Der Speicher, in dem die Bibliothek die zu versendenden Nachrichten akkumuliert, wird wiederverwendet. Da es geschehen kann, dass der Speicher nahezu gefüllt ist und eine weitere ausgehende Nachricht von der Anwendung eintrifft, deren Größe die verbleibende Kapazität überschreitet, wird ein zweiter Sendepuffer benötigt. Durch diesen Ansatz ist auch der Fall abgedeckt, dass der Puffer gerade durch den IP-Stack belegt ist, während weitere zu versendende Nachrichten eintreffen. Bei dem hier gezeigten Konzept des Wechselpuffers handelt es sich im wesentlichen um eine Ring-Queue mit 2 Elementen. Diese Queue wird oftmals verwendet um Erzeuger-Verbraucher-Probleme zu lösen.

Das im nächsten Abschnitt vorgestellte Protokoll besteht (Wie auch schon am Ende von Kapitel 5.2 kurz beschrieben) grundsätzlich aus zwei Phasen. Innerhalb der Initialisierungsphase wird eine Benutzerauthentifizierung durchgeführt. Diese Benutzerauthentifizierung sieht vor, zunächst zu prüfen, ob das System durch einen anderen Client in Benutzung ist. In diesem Fall wird mit einem Benutzer kommuniziert, um diesen abzuweisen. Dies geschieht alles während der Initialisierungsphase. Die Protokollverarbeitungseinheit ist nur für die Kommunikation mit dem verbundenen und authentifizierten Benutzer zuständig.

Damit die Nachrichten der Arbeitsphase zu Nachrichtenpaketen zusammengeführt werden können, müssen sie ein einheitliches Format besitzen.

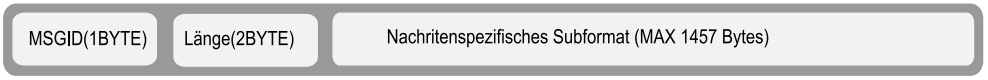


Abbildung 5.24: Generelles Format für Nachrichten der Arbeitsphase

Vorteile dieses Ansatzes

Die Zusammenführung von Nachrichten zu Nachrichtenpakete hat einen positiven Einfluss auf das Protokoll. Der Vorteil dieses Ansatzes ist, dass die zur Anbindung des Stacks nötigen Socketoperationen nie wieder angepasst werden müssen. Indem das Protokoll auf diese Ebene heraufgeholt wird, ist eine spätere Erweiterung des Protokolls um eigene Nachrichten sehr einfach. Das Auftreten von hinzugekommenen kleinen Nachrichten hat außerdem keinen weiteren Einfluss auf den Datendurchsatz, da es nicht vorkommt, dass kleine Pakete verschickt werden, die daraufhin die Sendekapazität herabsetzen.

Die Erzeugung und Auswertung der Nachrichten wird auf beiden Seiten der Verbindung von einheitlichen Mechanismen übernommen. Das Hinzufügen von neuen Nachrichten zum Protokoll bedeutet nur, dass an zentraler Stelle die Fallunterscheidungen für die Nachrichtenauswertung und Weiterleitung zu entsprechenden Komponenten hinzugefügt werden muß.

Beim Hinzufügen von Nachrichten muß allerdings das zugrundeliegende Basisformat (siehe 5.24) strengstens einhalten werden. Sobald die Headerinformationen des Grundformats eines Pakets zerstört werden, können die restlichen Nachrichten in diesem Nachrichtenpaket nicht mehr ausgewertet werden.

Die Grundsätzliche Erweiterbarkeit des Protokolls

Das Protokoll dieser Bibliothek ist sehr erweiterbar, da die zur Auswertung der Nachrichten verwendete Software jedes Mal von der Clientplattform geladen wird. Ein Entwickler, der das Protokoll um einen weiteren Befehl erweitert hat, muß nicht auf irgendwelche Protokollversionen achten. Da der Client für dieses Protokoll von der Mikroprozessorplattform geladen wird, hat der Benutzer immer eine kompatible Software zum System.

5.3.8 Web Server

Speicherung des Applets

Das Applet muß zunächst in den Flash-Speicher des Mikroprozessors gelangen. Hierzu gibt es tools, welche die Hex-Codes der einzelnen Bytes in einer C-Array

Notation ausgeben. Die entstandene Textdatei per `#include` in ein C-Projekt einbezogen werden. Eine `const`- Anweisung hilft dem Compiler, das Array im Programmspeicher zu belassen.

Auslieferung und Basiskonfiguration des Applets

Um das Applet auszuliefern, ist lediglich auf Port 80 ein Modul anzubinden, welches HTTP spricht. Es kann ein Webserver verwendet werden oder (um Programmspeicher zu sparen) ein weitestgehend statischer Server implementiert werden, welcher lediglich zwei Dateien ausliefert. Das Applet kann nur über eine Webseite betrachtet werden, welche das Applet einbettet. Diese Webseite benötigt technisch nichts weiter, als einen weitestgehend leeren Rumpf, der nur die `<applet>`-Tags zum Einbetten des Applets enthält. In den Embed-Parametern des Applets ist es möglich, unkritische Konfigurationsdaten zur Übergabe an das Applet zu platzieren. Ein Server-Modul, welches das Applet in Abhängigkeit der Konfigurationsdaten der Bibliothek bei der Auslieferung „dekoriert“, bietet die Möglichkeit, dem Applet gewisse Grundeinstellungen mitzugeben. Hierzu kann beispielsweise das Deaktivieren der Loggingeinheit in der Oberfläche zählen, wenn der Einsatzfall der Bibliothek diese Komponente nicht vorsieht. Die Oberfläche kann trotzdem mit Daten versorgt werden, um eventuell Bedienelemente, die eine Sensitivitätsliste verwenden mit Updates zu versorgen.

HTTP ETAG

HTTP ETAG[16] ist eine Technik, die das erneute Übertragen beliebiger Dateien verhindert, die sich bereits im Browser-Cache befinden. Damit das Applet nicht bei jeder Verbindungsaufnahme wieder ausgeliefert werden muss, obwohl sich nichts geändert hat, müsste der Webserver das ETag-Header-Feld im HTTP-Header verwenden. Bei der ersten Anfrage einer Ressource sendet der Server einen für diese Ressource spezifischen ETag-Wert im ETag-Header-Feld, der vom Client zusammen mit der Ressource lokal gespeichert wird. Bei einer erneuten Anfrage derselben Ressource sendet der Client in dem Header-Feld If-None-Match den zuvor gespeicherten ETag-Wert mit. Auf der Server-Seite wird nun der gesendete ETag-Werte mit dem aktuellen verglichen und bei Übereinstimmung mit dem Statuscode 304 beantwortet. Die Daten der Ressource werden in diesem Fall nicht mitgeschickt und der Client verwendet die lokal gespeicherten Daten.

[16] `http://de.wikipedia.org/wiki/HTTP_ETag`

Abwicklung des Anwendungsprotokolls

Für die Abwicklung des Anwendungsprotokolls ist ein weiteres Servermodul notwendig, welcher auf einem anderen Port als dem des Web-Servers arbeiten sollte. Das Modul muß zunächst die Benutzerauthentifizierung durchführen, die Initialisierungsphase des Protokolls einleiten (siehe 5.4) und die danach eintreffenden Daten an die im vorigen Abschnitt vorgestellte Protokollverarbeitungseinheit weiterleiten und ein Client-Connected-Event im Eventsystem auslösen.

5.4 Das Protokoll

Das verwendete Protokoll besitzt eine Initialisierungsphase. Die Initialisierung besteht aus drei Stufen. In der Verbindungsphase verbindet sich der Client. Der Server prüft, ob das System gerade belegt ist. Ist das System belegt, antwortet er mit der OCCUPIED_MSG und trennt die Verbindung. Ist das System frei und es ist eine Benutzerthentifizierung vorgesehen, wird die Authorisationsphase eingeleitet. Ist die Authorisationsphase erfolgreich, erfolgt die Katalogphase. In der Katalogphase werden die Datensätze, die das Datenmodell beschreiben, gesendet. Nach Abschluss der Katalogphase bekommt die Anwendung auf dem Controller ein USER-Connected-Event über das Event-System. Abbildung 6.8 zeigt die Initialisierungsphase des Protokolls. Die genauen Nachrichtenformate sind am Ende dieses Kapitels aufgeführt.

5.4.1 Initialisierungsphase

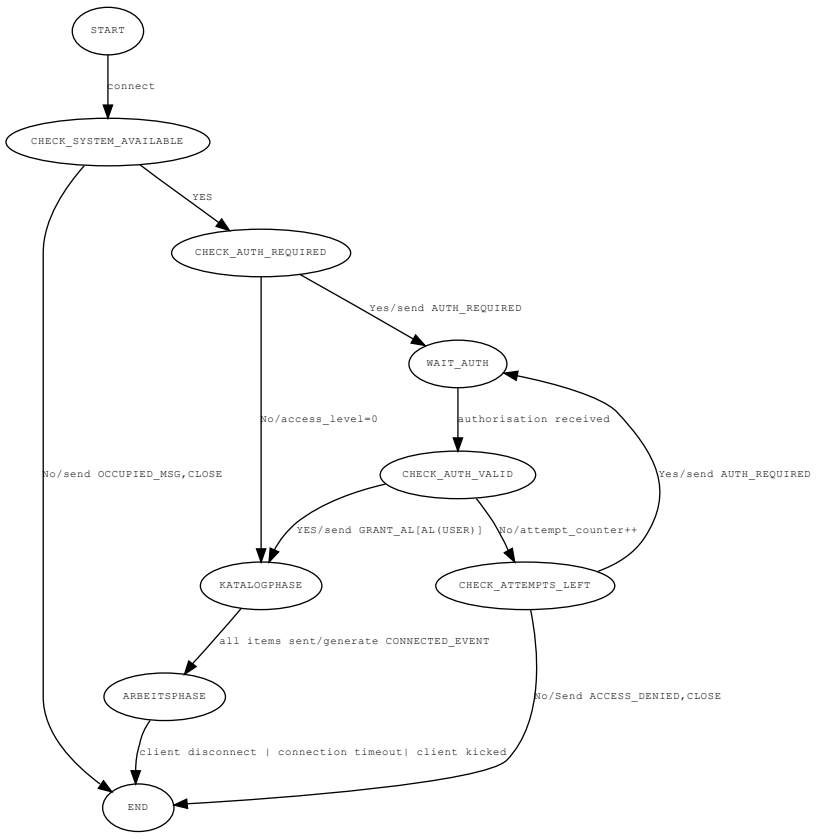


Abbildung 5.25: Protokoll-Automat (Initialisierungsphase)

Authentifizierungsphase

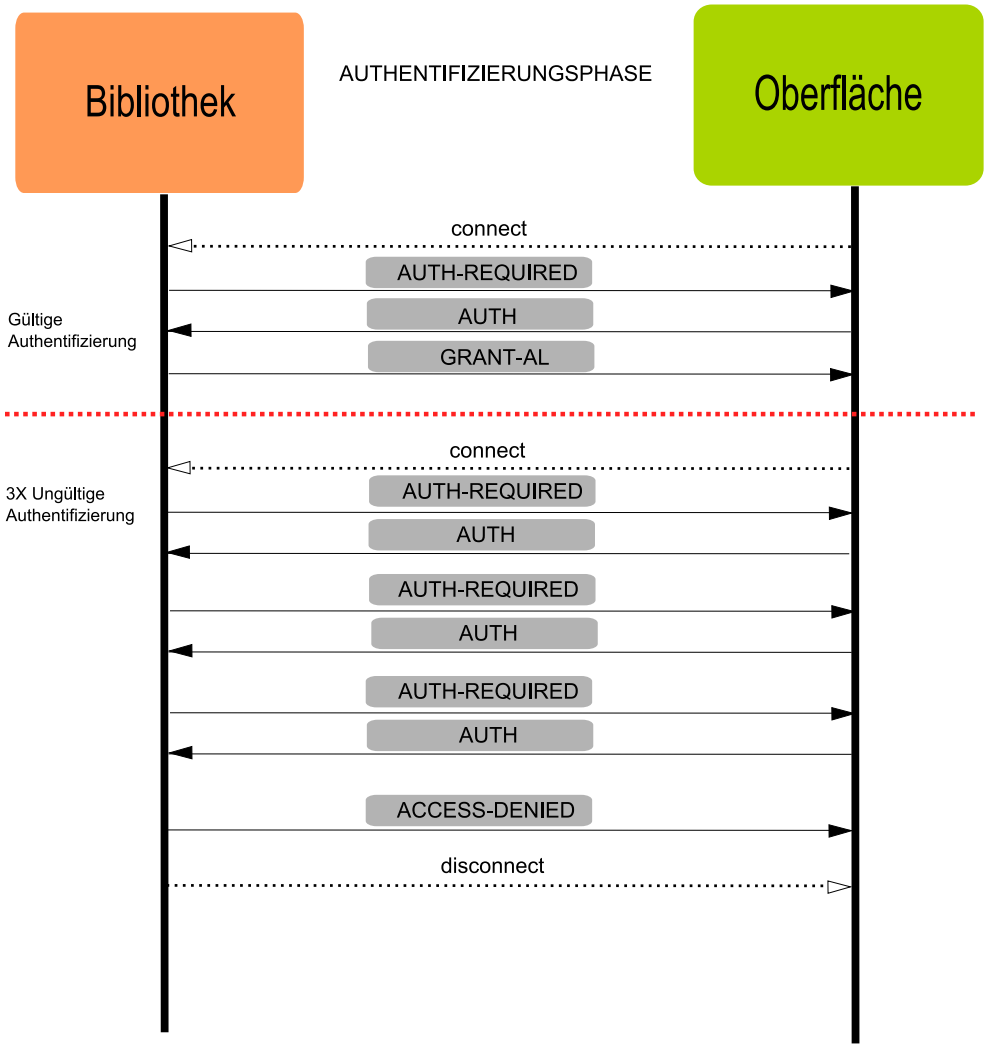


Abbildung 5.26: Die Authentifizierungsphase

Katalogphase

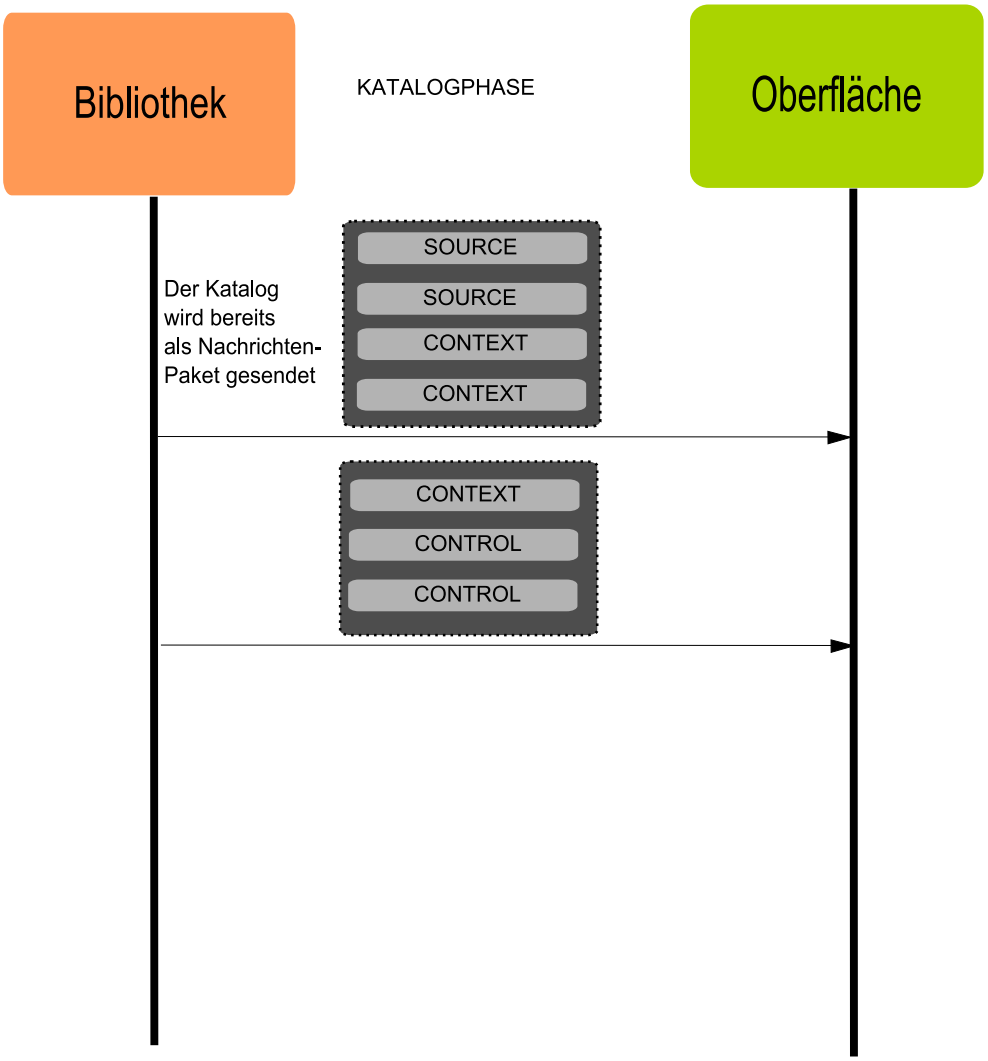


Abbildung 5.27: Die Katalogphase

Die Nachrichten der Katalogphase

Die Katalogphase dient der Initialisierung des Applets mit dem Datenmodell (Auch Katalog genannt) der konkreten Anwendung. Zur Validierung des Modells gibt es eine Validierungsroutine, die die Gültigkeit des Modells prüft (siehe 5.2. Nach einem Reset des Mikrprozessors wird bei der Initialisierung der Bibliothek das Datenmodell geprüft. Hat der Benutzer der Bibliothek ein ungültiges Modell beschrieben, antwortet die Bibliothek mit einem Aufschlussreichen Fehlercode und versagt den Dienst. Ist das Modell gültig, werden Katalogbeschreibungen übertragen. Die Nachrichten der Katalogphase erfüllen das in Abbildung 5.24 auf Seite 68 festgelegte notwendige Basisformat für alle

Nachrichten, die in Nachrichtenpaketen versendet werden.

Katalogphase

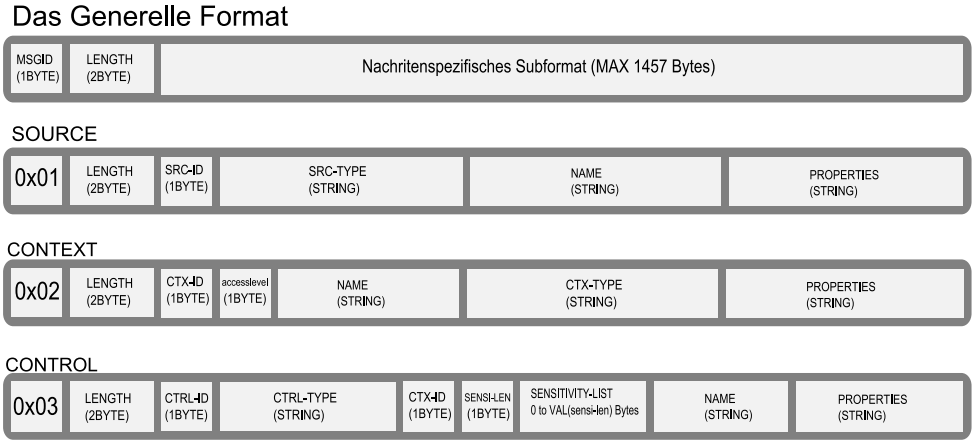


Abbildung 5.28: Die Nachrichten der Katalogphase

Die Felder die mit String benannt sind, sind nullterminierte C-Strings und haben eine variable Breite. Die Validierungsroutinen prüfen das Modell darauf, das folgende Bedingungen stimmen:

Die Längen der Strings gelten inklusive der terminierenden Null.

- **SOURCE**: Die gemeinsame Länge der Felder `SRC-TYPE`, `NAME` und `PROPERTIES` muss <= 1456 Bytes sein
- **CONTEXT**: Die gemeinsame Länge der Felder `CTX-TYPE`, `NAME` und `PROPERTIES` muss <= 1455 Bytes sein
- **CONTROL**: Die gemeinsame Länge der Felder `CONTROL-TYPE`, `NAME` und `PROPERTIES`, plus `SENSI-LEN` muss <= 1454 Bytes sein

5.4.2 Arbeitsphase

Die wichtigsten Nachrichten dieser Anwendung sind CONTROL-EVENT, CONTROL-STATUS und SAMPLE. Die Nachricht SAMPLE überträgt die Messdaten registrierter Push- und Pull-Quellen. Die Nachricht CONTROL-EVENT signalisiert ein Oberflächenereignis einer Bedienkomponente an den Mikroprozessor. Dieses Ereignis wird in das Ereignissystem der Anwendung eingespeist. Im Datenfeld sind die für den jeweiligen Bedienkomponententyp spezifizierten Daten enthalten. Ein Knopf kann beispielsweise einfach ein Byte mit dem Wert `0x01` für gedrückt und `0x02` für losgelassen senden. Ob der Knopf auch eine Nachricht für losgelassen sendet, würde beispielsweise in den Properties-Daten der Bedienkomponente im Datenmodell angegeben werden können. Mit der Nachricht CONTROL-STATUS signalisiert der Mikroprozessor einer Bedienkomponente einen Zustand. Zustandsinformationen können beliebige Daten enthalten, wie zum Beispiel Text für interaktive Hilfesysteme oder ähnliches. Letztendlich ist es möglich ein eigenes Subprotokoll für die Kommunikation mit der Oberflächenkomponente zu entwerfen. Die Nachrichten an Komponenten werden den Komponenteninstanzen in der Java-Software übergeben. Damit hat die Java-Software ihren Dienst auch schon getan.

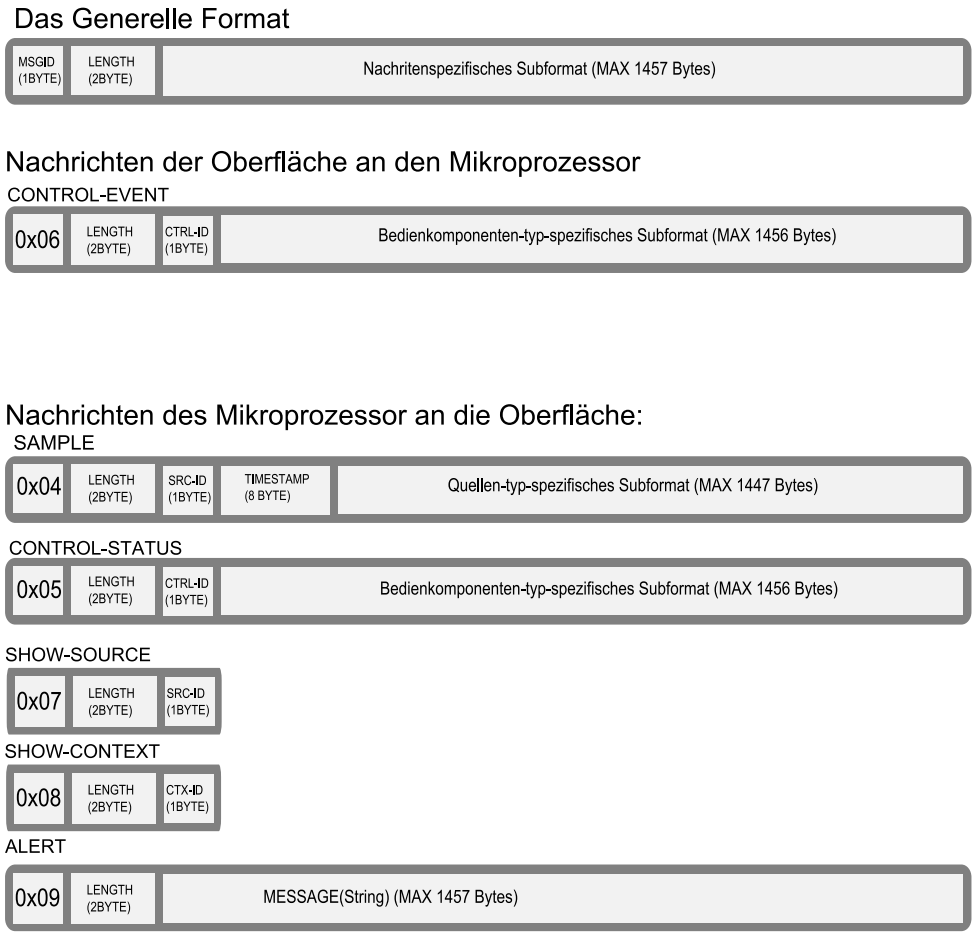


Abbildung 5.29: Die Nachrichten der Arbeitsphase

5.5 Die grafische Oberfläche

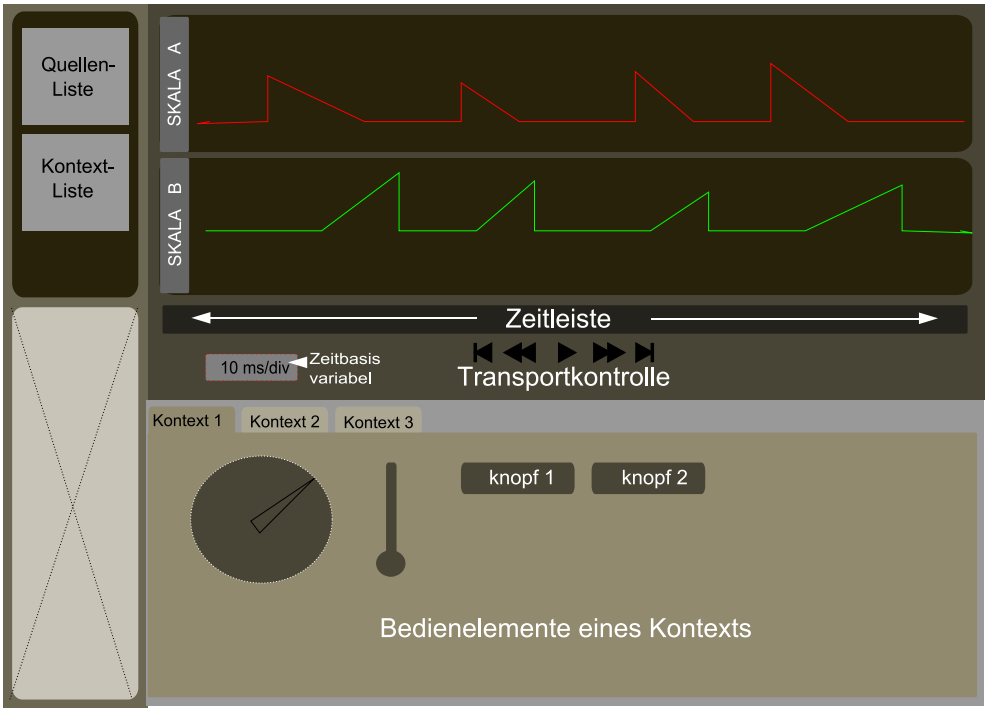


Abbildung 5.30: Entwurf der Oberfläche

Das in Kapitel 5.1 vorgestellte Oberflächenkonzept basiert auf dem Messquellen-Bereich, dem Kontext-Bereich und einer Sidebar. Damit die Oberfläche auch auf kleineren Displays gut funktioniert, ist vorgesehen, dass die Sidebar sich ausblenden lässt. Ebenso sollte der Messquellen-Bereich (im Folgenden auch Log-Viewer genannt) sich ausblenden lassen, falls eine Anwendung diese Komponente nicht braucht. Ebenso ist vorstellbar, dass der Kontextbereich ausgeblendet werden kann, falls die Bibliothek ausschlisslich zur Visualisierung verwendet werden soll. Auf diesem Wege ist der vielseitigen Verwendung der Oberfläche für unterschiedlichste Einsatzgebiete Sorge getragen. Diese Grundeinstellungen lassen sich bereits bei der Auslieferung des Applets über die `<param>`-Tags des Applets einstellen.

5.5.1 Verknüpfung von Datentypen

Das Datenmodell der Oberfläche besteht direkt aus den von der Mikroprozessorseite empfangenen Katalogdaten. Die in Abbildung 5.28 vorgestellten Katalogdaten beinhalten darüber hinaus alles, was zur Initialisierung der durch den Benutzer erweiterbaren Darstellungs- Kontext- und Bedienkomponententypen notwendig ist. In den jeweiligen Typ-Feldern gibt der Benutzer der Bibliothek die Namen der Java-Klassen an, die auf der Applet-Seite instanziert werden sollen. Hierfür wird auf Java-Seite eine Klasse benötigt, welche im Package

des Applets an geeigneter Stelle nach Klassen sucht und diese dann über die Mittel der Java *Reflection*[20] instanzieren kann. Auf diesem Wege muss keine weitere Klasse geändert werden, wenn neue Typ-Klassen hinzukommen. Bei der Instanzierung der Typ-Klassen dienen die aus den Properties-Feldern automatisch generierten Properties-Objekte als Parametrisierung. Die Parametrisierung kann bei dem Entwurf neuer Typ-Klassen verwendet werden, beliebige Informationen an das jeweilige Oberflächenelement zu übertragen. Die Notation in key/value-Paaren ist im Web-Bereich eine gängige Praxis. Bei der Notation im C-Sourcecode der Anwendung auf dem Mikroprozessor entsteht ein guter Überblick darüber, wie die jeweiligen Typ-Klassen parametrisiert sind.

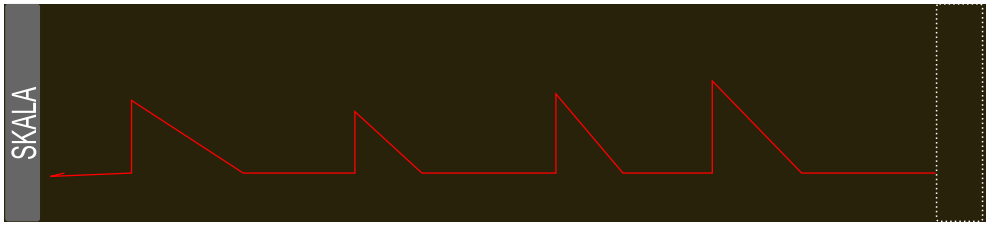


Abbildung 5.31: Darstellungsfläche einer Darstellungsklasse für Messquellen

5.5.2 Darstellungsklassen und Log-Viewer

Das Applet empfängt regelmäßig SAMPLE Nachrichten. Damit der Log-Viewer den gesamten Zeitbereich seit Verbinden des Applets darstellen kann, müssen alle SAMPLE Nachrichten aufgezeichnet werden. Die Zeitstempel können in gängigen *Container*-Klassen als Schlüssel für den sequenziellen Zugriff auf Sample-Nachrichten einer Messquelle dienen. Zu jeder Messquelle wird auf Java-Seite eine Aufzeichnug angelegt, sowie eine Darstellungsklasse instanziert, welche das Properties-Objekt als Parametrisierung erhält. Die Darstellungsklassen dienen der Darstellung im Log-Viewer. Sobald eine Messquelle im Log-Viewer angezeigt werden soll, wird ihre Darstellungsklasse beim Log-Viewer registriert. Die Darstellungsklasse hat Zugriff auf die Daten ihrer Messquelle. Die Darstellungsklassen benötigen einen Mechanismus, über den ihnen der zeitliche Abschnitt, der angezeigt werden soll, durch den Log-Viewer mitgeteilt werden kann. Darstellungsklassen können sehr frei entworfen werden. Das Zeitfenster, welches dargestellt werden soll, könnte beliebig visualisiert werden. Der Umgang mit Java2D[18] ist sehr einfach. Das Basiskonzept bei der Darstellung von Messquellen im Log-Viewer ist, dass die Komponenten keine eigene Zeitleiste zeichnen, wohl aber ihre eigene Skala. Beim Zeichnen von beispielsweise einem Funktionsplot muss die Darstellungsklasse zunächst alle im dargestellten Zeitfenster enthaltenen Messdaten aus der Aufzeichnung holen. Die Position der Daten auf der X-Achse lässt sich durch einfache Berechnungen mit den enthaltenen Zeitstempeln ermitteln. Da die Darstellungsklassen ihre eigene Skala zeichnen ist es möglich, Darstellungsklassen zu entwickeln, welche über Auto-Range Funktionen verfügen. Nicht alle Displayklassen be-

nötigen eine Skala. Genauso wenig muß die Skala auf der linken Seite sein. Die benötigten Displayflächen für Skalen auf der Linken und der rechten Seite der Darstellungsfläche sollten beim Log-Viewer bekannt gemacht werden. Dieser kann daraufhin allen anderen Darstellungsklassen Offsets bekannt machen, damit die im Log-Viewer dargestellten Darstellungsklassen weiterhin alles untereinander zeichnen. Die Implementierung eigener Displayklassen muß deswegen nicht sonderlich komplex sein. Die Bereitstellung geeigneter abstrakter Basisklassen würde die notwendige Funktionalität weitestgehend bereitstellen und dem Entwickler neuer Klassen nur das Zeichnen überlassen, sobald sich etwas ändert.

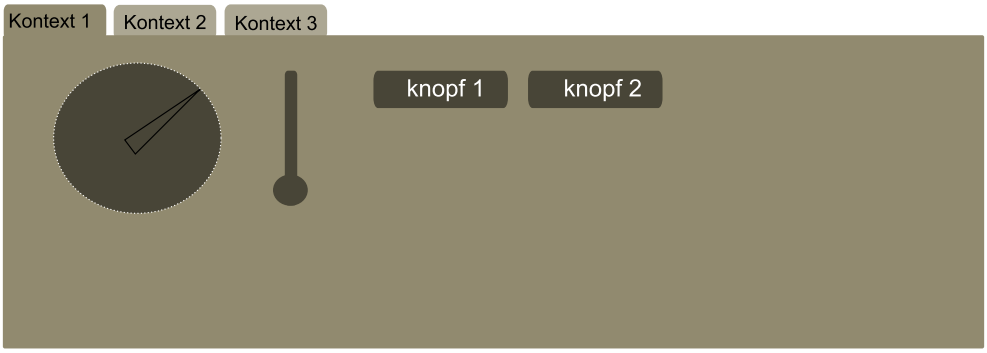


Abbildung 5.32: Bedienkontexte und Bedienelemente

5.5.3 Kontexte

Kontexte enthalten Bedienelemente. Die Bedienelemente werden nach der Katalogphase über spezifizierte Methoden an die Kontexte übergeben. Die Bedienelemente sollten natürlich auch angeordnet werden können. Java bietet hierzu eine Vielzahl von sinvollen Layoutmanagern[19] welche helfen können, die Anordnung von Bedienkomponenten zu erleichtern. Sogenannte Null-Layouts ermöglichen die genaue Platzierung aller Komponenten. Hierfür ist sicherlich etwas Fingerspitzengefühl erforderlich. Es ist allerdings auch denkbar, einen Kontext auf Basis des Null-Layouts zu entwerfen, welcher über einen Edit-Modus verfügt. Die enthaltenen Bedienelemente sind über die Katalogdaten eindeutig identifiziert. Diese Informationen kann ein Kontext benutzen, um Darstellungsparameter für enthaltene Bedienelemente zu formulieren, welche dann in Properties-kompatibler Form wieder in die C-Software eingebracht werden können. Weitere Layoutmanager könnten folgen.

5.5.4 Bedienelemente

Bedienelemente müssen von geeigneten Klassen abgeleitet werden und für die Kommunikation mit dem Eventsystem *Interfaces* implementieren. Ein Bedienkomponenten-Typ ist nicht sehr komplex. Damit er funktioniert muß er lediglich eine Me-

thode besitzen, über die er mit eintreffenden CONTROL-STATUS Nachrichten versorgt werden kann. Damit der Bedienkomponenten-Typ CONTROL-EVENT Nachrichten verschicken kann, genügt globaler Sendemechanismus. Das genaue Format der Nachrichten kann durch den Entwickler einer Bedienkomponente selbst festgelegt werden. Durch *Swing*[21] steht eine gigantische Palette an Bedienelementen zur Verfügung. Das Konzept von Swing ist sehr durchdacht und an vielen Stellen sehr einfach gelöst. Viele Swing-Objekte können auf einfachstem Wege zu Bedienkomponententypen gemacht werden, indem einfach davon abgeleitet und das nötige Interface für die Eventverarbeitung implementiert wird. Komplexere Bedienkomponenten können mit den üblichen Mitteln der Swing-Programmierung erzeugt werden.

6 Realisierung

Für den Entwurf der Bibliothek ist ein Mikroprozessorboard mit Ethernet-Unterstützung notwendig. Das in dieser Arbeit verwendete Entwicklungsoard ist mit einem ARM7TDMI, dem LPC2468[33] von NXP[1] bestückt. Der Prozessor und der Ethernetcontroller werden in Anhang A und B genauer beschrieben. Für den Betrieb des Ethernetcontrollers ist ein IP-Stack notwendig, welcher sich bei der Kommunikation mit dem Netzwerk um die Verarbeitung der unteren Protokollebenen kümmert. Im folgenden Kapitel wird zunächst die Auswahl eines IP-Stacks getroffen.

6.1 Auswahl eines IP-Stacks

6.1.1 Verfügbare IP-Stacks im Vergleich

uIP

Der TCP/IP stack „uIP“[2] (micro IP) wurde entwickelt, um TCP/IP-Kommunikation auf 8-Bit-Microcontrollern zu ermöglichen. Der leichtgewichtige Stack mit einem Footprint von ein paar hundert Bytes erfüllt dabei alle Anforderungen, die Host-Zu-Host-Kommunikation benötigt. Um die Codegröße zu minimieren, sind verschiedene Mechanismen in der Schnittstelle zwischen der Anwendung und dem Stack entfernt worden, wie zum Beispiel das „Soft Error Reporting“ und die dynamisch konfigurierbaren „type-of-service“ bits für TCP Verbindungen. Die Schnittstelle ist nicht etwa mit einem vollwertigen BSD-Socket zu vergleichen, die Anwendung wird vielmehr durch den Stack aufgerufen, wenn ein Netzwerkereignis stattfindet. Anwendungen, die mit uIP kommunizieren werden normalerweise als Automaten implementiert, die durch den Stack mit Ereignissen versorgt werden. Der Stack muss regelmäßig aufgerufen werden, dies kann in einem Main-Loop oder als Task in einem Echtzeit-Betriebssystem stattfinden. Da uIP nur einen einzigen globalen Puffer für eingehende und ausgehende Nachrichten verwendet, sind an dieser Stelle nur einige wenige Operationen notwendig, bevor das Paket der Anwendung übergeben wird. Es steht der Anwendung frei, das Paket zu kopieren oder die Informationen an Ort und Stelle auszuwerten. Der Stack verzichtet auf das

[1] www.nxp.com
[2] http://www.sics.se/~adam/uip/index.php/Main_Page

Verfahren des „Sliding Window“(Siehe hierzu Kapitel 3.3.2), stattdessen ist immer nur genau ein Paket auf dem Weg zum Client, bis dieses Paket bestätigt wurde. Dieses zutiefst in uIP verankerte Verhalten ist für den kleinen Footprint des Stacks verantwortlich und hat einen großen Einfluß auf die Anwendungsschnittstelle des Stacks. Nur so ist es möglich, den Stack portabel zu halten und selbst auf Systemen mit sehr wenig Speicher auszuführen. Der Stack kümmert sich zwar um die ARP-Schicht und den Verbindungsaufbau, das Abarbeiten des Paketversands geschieht jedoch in enger Zusammenarbeit mit der Anwendung. Um Speicher zu sparen, wird nicht einmal das zuletzt versendete Paket gespeichert, bis dieses bestätigt wird. Stattdessen benachrichtigt der Stack im Falle eines nicht erfolgten Ack-Pakets die Anwendung, die daraufhin das zuletzt versendete Paket über die übliche Senderoutine verschickt. Dies hat nicht unbedingt einen großen Einfluß auf die Komplexität der Anwendung, da sowieso nur ein Paket zur Zeit verschickt wird. Wie die Ereignisse letztendlich verarbeitet werden, liegt in der Hand des Entwicklers. Da sich durch das ereignisgesteuerte Konzept des Stacks eine Implementierung der Anwendung als Automat geradezu unumgänglich macht, liefert der Stack die Möglichkeit, beliebige Zustandsinformationen des Automaten zu einer Verbindung zu speichern. Diese werden zur Compile-Zeit festgelegt. So ist einfach möglich, einen Automaten zu implementieren, der gleichzeitig beliebig viele Verbindungen versorgen kann. uIP ist sehr skalierbar. Zur Compilezeit kann die Anzahl der maximalen gleichzeitigen Verbindungen konfiguriert werden. Auch einige nicht unbedingt notwendige Zusatzfunktionen können durch einfache `#define`-Anweisungen abgeschaltet werden. Der bei kleinen Prozessoren notwendige Overhead der Verarbeitung von TCP/IP-Nachrichten wird von den notwendigen Kopieroperationen und Checksummenberechnungen dominiert.

lwIP

Der TCP/IP stack „lwIP“ [3] (lightweight IP) ist eine Erweiterung des uIP Stacks, welcher eine komplette TCP/IP Implementierung bereitstellt. Aus diesem Grunde ist ein direkter Vergleich von uIP und lwIP an dieser Stelle interessant. Abbildung 6.1 zeigt einen kurzen Überblick. Es gibt zwei verschiedene Arten, mit dem Stack zu arbeiten. Entweder können die einzelnen Funktionen in den zugrundeliegenden TCP- und UDP-Modulen direkt aufgerufen werden oder es kann Verwendung von der Berkeley-Socket-artigen „lwIP-API“[15] gemacht werden. Die Schnittstelle der lwIP-API basiert wie bei uIP auf *callbacks*. Die Anwendungschnittstelle von lwIP unterscheidet sich allerdings von der von uIP, da lwIP sich im Gegensatz zu uiP selbst um die Neuübertragung von verlorengegangenen Paketen kümmert. Dies hat allerdings auch einen großen Einfluß auf den Speicherbedarf (RAM und ROM) von lwIP.

[3] `http://savannah.nongnu.org/projects/lwip/`

Feature	uIP	lwIP
IP and TCP checksums	x	x
IP fragment reassembly	x	x
IP options		
Multiple interfaces		x
UDP		x
Multiple TCP connections	x	x
TCP options	x	x
Variable TCP MSS	x	x
RTT estimation	x	x
TCP flow control	x	x
Sliding TCP window		x
TCP congestion control	Not needed	x
Out-of-sequence TCP data		x
TCP urgent data	x	x
Data buffered for rexmit		x

Abbildung 6.1: Vergleich von *uIP* und *lwIP* Quelle: Adam Dunkels[16]

lwIP bietet das Routing von Paketen über mehrere Netzwerkschnittstellen und Elemente des ICMP-Protokolls für erweiterte Netzwerkfunktionen. Der IP-Layer in lwIP ist außerdem angewachsen, da lwIP UDP Broadcast- und Multicastpakete, sowie die dazugehörigen ICMP UDP-error Nachrichten unterstützt. Die Implementierung von lwIP benötigt vier mal so viel ROM wie uIP, da lwIP im Gegensatz zu uIP den *Sliding Window*-Mechanismus implementiert, welcher verhältnismäßig viel Speicher und diverse Queue-Management-Funktionen benötigt. Es gibt viele Gründe für den dramatischen Größenunterschied von uIP und lwIP. Um komplexere TCP implementationen zu ermöglichen, hat lwIP eine komplexere Puffer- und Speicherverwaltung als uIP. Die unterschiedlichen Ansätze von lwIP und uIP haben einen Einfluss auf die Codegröße. Die Unterstützung für dynamisch wechselnde Netzwerkschnittstellen in lwIP ist außerdem für den erhöhten Speicherbedarf verantwortlich, da der IP-Layer mehrere lokale IP-Adressen verwalten kann.

NicheLite™

Der TCP/IP stack „NicheLite“[4] der Firma InterNiche

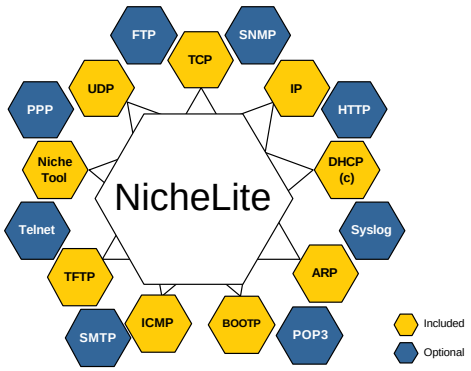


Abbildung 6.2: Konzept des IP-Stacks NicheLite™ Quelle: InterNiche

NicheLite™ ist laut Angaben der Firma InterNiche ein voll ausgebauter TCP/IP-Stack, der nur 12 KB speicher benötigt. NicheLite bietet die sogenannte „Mini“ Sockets API. Der IP-Layer unterstützt eine Netzwerkschnittstelle und ist voll kompatibel zu weiteren Produkten von InterNiche. InterNiche bietet neben dem Stack auch FTP, Telnet-, HTTP-, DHCP- und SNMP-Server als optionale Module an. Hierbei bietet NicheLite MIB-II SNMP-Zugriff auf die MIB-II Interface-, IP-, ICMP-, TCP- und UDP-Tabellen. NicheLite wird als klein, schnell und für eingebettete Systeme optimiert beworben. Um das Kopieren von Datenpaketen zu vermeiden und die Speicherverwaltung zu vereinfachen, werden Paketdaten in aneinandergrenzenden Puffern an die Anwendung ausgeliefert. Der Sliding-Window-Mechanismus steht zur Verfügung und kann von der Anwendung zur Laufzeit beeinflusst werden. Um die Portierung von NicheLite zu beschleunigen, liefert InterNiche die Software „NicheTool“ mit. Diese soll beim Debuggen und Feintunen des Stacks unterstützen. Für die Verwendung von NicheLite™ ist eine einmalige Zahlung zu entrichten. Für ausgelieferte Stückzahlen sind keine weiteren Zahlungen fällig.

*u*C/TCP-IP™

Der TCP/IP stack „*u*C/TCP-IP“[5] der Firma Micri*u*m ist ebenfalls modular aufgebaut, wobei weitere optionale Komponenten wie DHCP client, DNS client, TFTP client, TFTP server, FTP client, FTP server, HTTP server, SNTP client, POP3 client, SMTP client und TELNET server dazugekauft werden können. Der Stack unterstützt kein UDP-Multicast. Micri*u*m bewirbt bei seinem Produkt die hohe Qualität des Sourcecodes. *u*C/TCP-IP ist als sogenanntes „clean-room“ Design entstanden und bietet volle Kompatiblität zur Berke-

[4] `http://www.iniche.com/nichelite.php`
[5] `http://www.micrium.com/products/tcp-ip/tcp-ip.html`

ley 4.4 Socket Layer Interface Spezifikation. Auch uC/TCP-IP versucht in seinem Produkt, die Verarbeitung von Netzwerkpaketen zu beschleunigen, indem das Kopieren von Segmenten vermieden wird. Allerdings hat dieses Produkt von allen bisher erwähnten IP-Stacks den größten Footprint. Die beworbene Skalierbarkeit ist nur eingeschränkt vorhanden. Bei einem ROM-Footprint von 78 Kilobyte beträgt der Unterschied zwischen einer Version, bei der alle Optionen aktiviert sind und einer, bei der alle Optionen deaktiviert sind gerade einmal 4 Kilobyte. Auch für die Verwendung des IP-Stacks von Micri$u$m ist eine einmalige Zahlung zu entrichten. Für ausgelieferte Stückzahlen sind keine weiteren Zahlungen fällig.

Treck TCP/IP™

Der TCP/IP stack „Treck TCP/IP“[6] der Firma Treck wird ebenfalls als schnell beworben, da auch er auf das Kopieren von Paketen verzichtet. Eintreffende Daten werden laut Treck vom Treiber über den Stack bis in die Anwendung hinein nicht ein einziges Mal kopiert. Die Anwendungsschnittstelle dieses Stacks ist ebenfalls der eines Berkeley sockets nachempfunden. Treck bietet die typischen grundlegenden Protokolle für den Betrieb von Ethernet. Hierzu gehört auch UDP. Optional sind Module für PPP, NAT, IGMP, DHCP, FTP, TFTP, SNMP, Telnet, Mobile IP, IPsec, DNS Resolver, HTTP, POP3 und SMTP erhältlich. Die Device-Driver-Api wird als besonders einfach beworben. Viele Konfigurationsoptionen sind zur Laufzeit verfügbar, laut Treck ist sogar das Hinzufügen von Treibern zur Laufzeit ist möglich. Treck bietet weitere IP-Stacks mit unterschiedlichen Schwerpunkten an, hierzu gehören zum Beispiel Produkte für SSL, IPv6, IPsec, Mobile IP und Mobile IPv6. Treck schreibt über seine Produktreihe, dass es für einen Integrator keinen Grund geben sollte, mit den Innereien ihrer Produkte vertraut zu sein. Treck bietet „saubere“ Schnittstellen für den Treiber, die Sockets und verschiedene APIs an um ihre Produkte zu integrieren. Produkte von Treck verfügen laut [36] über ein eigenes Locking-System, was die Integration in preemptive Betriebssysteme erleichtern soll. Durch die Verwendung von zählenden Semaphoren wird laut Angaben von Treck der konkurrierende Zugriff auf gemeinsame Daten sichergestellt. Treck verwendet eine dynamische Speicherverwaltung, die bei Bedarf Speicher vom Betriebssystem anfordert. Der Speicherpool von Treck wird erweitert, falls die Speicheranforderungen zur Laufzeit ansteigen. Hierdurch sind laut Treck keine „Rätseleien“ beim Entwurf der Anwendung nötig, um den konkreten Speicherbedarf für den Stack zu ermitteln. Für die Verwendung des IP-Stacks von Treck ist ebenfalls eine einmalige Zahlung zu entrichten. Für ausgelieferte Stückzahlen sind keine weiteren Zahlungen fällig.

[6] `http://www.treck.com/`

Bewertung der IP-Stacks

Um einen Vergleich der verfügbaren IP-Stacks zu vereinfachen, zeigt Tabelle 6.1.1 eine kurze Übersicht über die Eigenschaften der einzelnen Produkte. An dieser Stelle ist bereits zu erkennen, dass die Opensource-Produkte neben den Kommerziellen Produkten durchaus mithalten können. Besonders bemerkenswert ist der geringe Footprint von uIP. Das Nichtvorhandensein einer UDP-Unterstützung ist an dieser Stelle kein Problem, da für die Kommunikation ausschließlich TCP benötigt wird.

	uIP	lwIP	NicheLite	uC/TCP-IP	Treck IP
ROM(kb)	5	22	12	78	32
RAM(kB)	< 3	< 10	< 10	54	< 20
Interfaces	1	Multiple	Multiple	Multiple	Multiple
TCP	JA	JA	JA	JA	JA
UDP	NEIN	JA	JA	kein Multicast	JA
uC-8bit	JA	JA	-	-	-
uC-16bit	JA	JA	JA	JA	JA
uC-32bit	JA	JA	JA	JA	JA
uC-64bit	Teilweise	Teilweise	Teilweise	Teilweise	Teilweise
Lizenz	BSD	BSD	kommerziell	kommerziell	kommerziell
Preis	kostenlos	kostenlos	royalty free	royalty free	royalty free

6.1.2 Hardwarelösungen

Es existieren auch Möglichkeiten, eine Anbindung ans Netzwerk über eigenständige Chips herzustellen. Die Palette reicht hier von TCP-fähige Bausteine bis zu embedded Webservern in der Größe einer RJ-45 Netzwerkbuchse. Im Folgenden werden zwei typische Vertreter dieser Bausteine vorgestellt, um einen Vergleich zwischen einer reinen Softwarelösung und einer kombinierten Hardware/Software-Lösung möglich zu machen.

Wiznet W3100A

Abbildung 6.3: Der W3100A von Wiznet™ Quelle: wiznet.co.kr

Bei dem Wiznet[7] W3100A handelt es um einen 10/100MBit Ethernet-Controller mit einem eingebauten TCP/IP-Stack. Er unterstützt hardwaremäßig die Protokolle TCP, IP, UDP, ICMP, ARP und MAC(PHY-Kommunikation). Der Baustein kann bis zu vier gleichzeitige TCP/IP-Verbindungen bedienen. Der Chip verfügt über 16KB internes RAM. Die Ansteuerung dieses Chips erfolgt über eine API, deren Source Code offen liegt und die leicht zu portieren ist. Die API ist laut dem Hersteller ähnlich zu Windows Sockets. Die Anbindung an einen Mikroprozessor erfolgt entweder über einen parallelen 8 Bit Datenbus oder über eine I2C - Schnittstelle. Der Chip wird mit 3,3 Volt betrieben und bietet 5V kompatibles IO. Neben dem W3100A ist weiterhin ein externer PHY-Chip notwendig. Preis: ca. 6 Dollar.

Lantronix X-Port

Abbildung 6.4: Der X-Port ©Lantronix

Der XPort® ist eine kompakte integrierte Lösung um Geräte mit serieller Schnittstelle mit dem Ethernet zu verbinden. Er wird mit einer Windowsbasierten Konfigurationssoftware geliefert, die die Installation vereinfachen soll. Der X-Port kann auch über den seriellen Port oder über einen Telnet-Zugang eingestellt werden. Er bietet eine 256-Bit AES Verschlüsselung für sichere Kommunikation. Er verfügt über einen eigenen Flash-Speicher, um Webseiten zu speichern. Der Chip wurde erfolgreich in einem Projekt[8] der Zeitschrift „C't“ eingesetzt. Durch den hohen Preis kann der X-Port allerdings schon an dieser Stelle ausgeschlossen werden, der Chip kostet ca 50 Dollar.

[7] `http://www.wiznet.co.kr`
[8] `http://www.heise.de/ct/04/13/200/default.shtml`

6.1.3 Bewertung

	Gewichtung	uIP	lwIP	NicheLite	uC/TCP-IP	Treck IP	W3100A	X-Port
Systemressourcen	5	5	3	4	1	2	3	5
Flexibilität	3	3	5	4	4	4	2	1
Entwicklungsaufwand	1	4	4	2	1	2	4	5
Stückkosten	4	5	5	4	1	4	3	1
Portabilität	3	5	5	4	3	3	4	5
Summe		**73**	**69**	**62**	**31**	**49**	**49**	**52**

Letztendlich scheint uIP sehr geeignet für dieses Anwendungsfeld zu sein. Durch den kleinen Footprint und die gute Skalierbarkeit sollte der IP-Stack der eigentlichen Mess- und Regelanwendung auch auf kleineren Plattformen noch genügend Ressourcen übrig lassen. Bei der Verwendung von uIP sollte das Delayed Acknowledgement beim Kommunikationspartner abgeschaltet werden. Da Java zum Einsatz kommt, ist dies sehr einfach. Beim Öffnen des Sockets wird einfach die Socketoption TCP_NO_DELAY angegeben. Gängige Browser bieten leider keine Möglichkeit, das Delayed Acknowledgement auszuschalten. Bei einem Client, der das Delayed Acknowledgement nicht in seinem Stack deaktiviert hat ist damit zu rechnen, dass die Übertragung des Java-Applets an den Client mit gebremster Geschwindigkeit ausgeführt wird. Bei einer Applet-Größe von 50 Kilobyte dauert dieser Vorgang dennoch nur 7 Sekunden. uIP ist der einzige Stack, der explizit auf 8-Bit Systemen unterstützt wird, wodurch die Palette an unterstützten Mikrocontrollern weiter wächst. Die nicht Berkeley-Socket kompatible Anwendungsschnittstelle ist gut durchdacht und dokumentiert.

Bei der Lizenz von μIP handelt es sich um die freie BSD-Lizenz. Software unter der BSD-Lizenz darf frei verwendet werden und es ist erlaubt, sie zu kopieren, zu verändern und zu verbreiten. In groben Zügen ähnelt das Lizenzmodell der GNU General Public License (GPL), nur dass es noch liberaler formuliert ist und kein Copyleft enthält. Die Lizenz eignet sich auch für kommerzielle Produkte. Ein Programmierer, der ein unter einer BSD-Lizenz veröffentlichtes Programm verändert und dann verbreitet, ist nicht verpflichtet, den Quellcode seines veränderten Programms mit zu veröffentlichen. Einzige Bedingung der BSD-Lizenz ist, dass der Copyright-Vermerk des ursprünglichen Programms nicht entfernt werden darf.]

6.2 Performancemessung mit uIP

Zunächst wurde ein simpler Benchmark implementiert. Das Ziel des Benchmarks war, die theoretischen Datenraten durch Messungen in der Realität zu bestätigen. Bei der Berechnung der theoretischen maximalen Datenrate (siehe Kapitel 3.3.2) müssen letztendlich die Zeiten summiert werden, die zwischen dem Senden von zwei aufeinanderfolgenden Paketen vergehen. Zu diesen Zeiten gehört auch die Rechenzeit der Anwendung, beziehungsweise die Rate, mit der die Stack-Routine aufgerufen wird. In einer Superloop-Architektur[28, S.164] werden alle für die Abwicklung einer Aufgabe notwendigen Funktionen mehr oder weniger in einer globalen Endlosschleife erledigt. Dies hat den Vorteil, dass eine Regelanwendung dadurch maximal schnell arbeitet. Beim Einsatz eines Schedulers wie zum Beispiel eines *Kooperativen*[28, S.246] Schedulers können zwar auf einem einfachem Wege zeitliche Garantien für gewisse Regelungsvorgänge gegeben werden, jedoch verwenden diese Schheduler oft regelmäßige Zeitslots, auf die die verschiedenen Aufgaben verteilt werden. Da diese Zeitslots etwas länger als die längste Task sein müssen, die durch den Scheduler aufgerufen werden soll, wird auf solchen Systemen bei kürzeren Tasks einige Rechenzeit verschwendet, um darauf zu warten, dass der Scheduler den nächsten Task ausführt. In einer Superloop-Architektur können Regelungsaufgaben durchgeführt werden, die mit gewissen periodischen Schwankungen leben können. Im Gegenzug wird allerdings die maximale Geschwindigkeit aus solchen Systemen herausgeholt.

Der Benchmark simuliert den wechselseitigen Aufruf einer Anwendung, und dem IP-Stack. Der Benchmark besteht aus einem Python-Testscript und einem kleinen Programm, das auf die Socket-API von uIP zugreift. Die Mikrocontroller-Seite kann zwei Dinge: Pakete verschicken und einen Wert vom Script empfangen, um eine Anwendungslast zu simulieren. Der Wert wird verwendet, um eine Zählschleife arbeiten zu lassen, deren einfache Aufgabe es ist, Rechenzeit zu verschwenden. Hierdurch wird die Anwendungslast simuliert. Das Skript sendet in regelmäßigen Abständen neue Last-Werte. Die simulierte Last hat einen Einfluss auf die maximale Datenrate des Stacks. Das Python-Testscript empfängt die Datenpakete, deren Größe der maximalen Größe eines Ethernet-Frames entspricht(1460 Bytes Daten + 40 Bytes Ethernet-Header), und berechnet aufgrund der eintreffenden Daten die jeweiligen Datenraten bei der jeweiligen Anwendungslast.

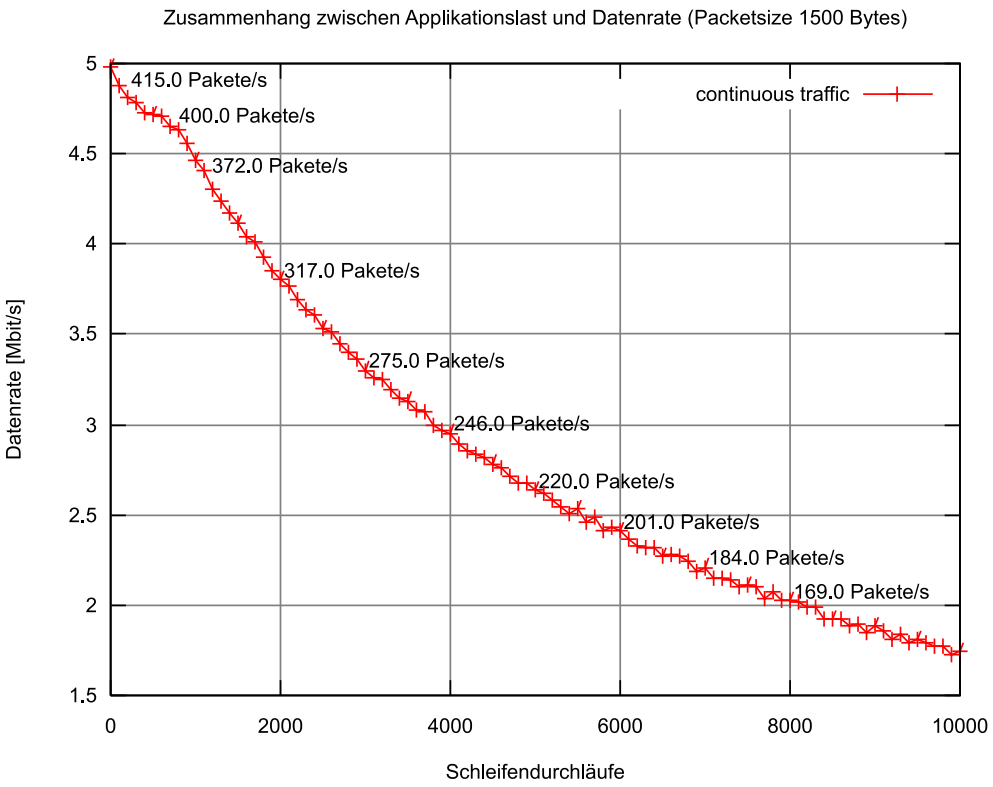


Abbildung 6.5: Benchmark mit uIP

Abbildung 6.5 zeigt das Ergebnis der Messung. Der Kurvenverlauf hat einen ähnlichen Funktionsverlauf wie die Funktion $f(x) = 1/x$, (nur das diese Funktion im Bereich um 0 bedauerlicherweise nicht gegen unendlich tendiert...). Die Funktion zur Berechung der Datenrate könnte also auch lauten(vgl s. **??**):

$$p = \frac{s}{t_r + t_d + t_a} \tag{6.1}$$

p Maximaler Datendurchsatz
s Segmentgröße in Bytes
t_r Round-Trip-Time RTT
t_d Delayed Acknowledge Timeout
t_a Maximale Zeit, die eine Anwendung in einem Aufruf benötigt

Bei der Verwendung von uIP ist der maximale Datendurchsatz des IP-Stacks davon abhängig, wie oft er aufgerufen wird. In einer Superloop-Architektur muss vom Worst-Case ausgegangen werden, also der Zeit, die im Extremfall zwischen den Aufrufen des Stacks vergeht. Dieser Benchmark wurde bei einer Round-Trip-Time von ca 2 ms durchgeführt. Bei der Konzeption einer Anwendung muss berücksichtigt werden, welches die zu erwartende maximale Round-Trip-Time ist. Soll das System per Fernwartung verwaltet werden,

sollte vorher ermittelt werden, welche Zeiten im schlechtesten Fall zu erwarten sind. Das *Delayed Acknowledge* muß bei der Verwendung der Bibliothek nicht berücksichtigt werden, da es durch den Einsatz von Java möglich ist, dieses Verhalten für den verwendeten Socket des Applets zu deaktivieren.

6.3 Aufbau der Mikroprozessor-Bibliothek

Da der zeitliche Rahmen dieser Arbeit sehr begrenzt ist, wurde die Software nicht in vollem Umfang implementiert. Die entstandenen Komponenten sind weitestgehend so implementiert worden, wie in Kapitel 5 beschrieben. Die C-Software wurde auf zwei Module aufgeteilt:

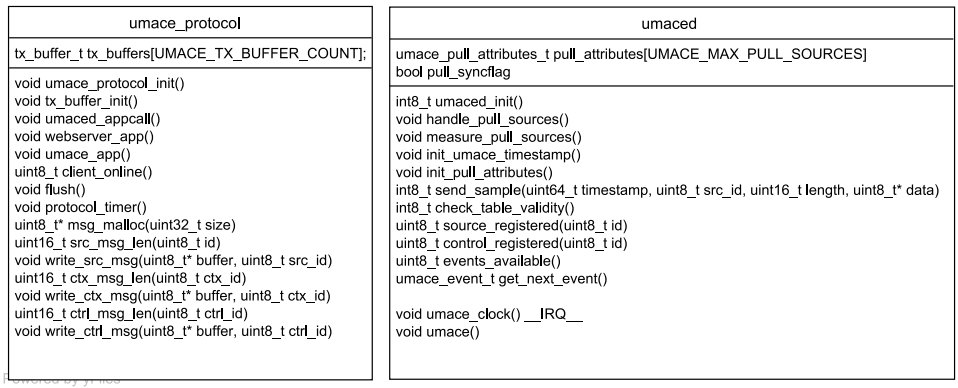


Abbildung 6.6: Die C-Bibliothek

Das Modul `umaced` enthält die Zeitstempel-Uhr, die Mechanismen zur Verarbeitung von Pull-Quellen und das Eventsystem. Das Modul `umace_protocol` enthält die Protokollverarbeitung bedient die Socket-Schnittstelle von uIP. Die Socket-Schnittstelle von uIP ist in [17] beschrieben und basiert auf einem Callback-Mechanismus. Der Stack muss regelmäßig aufgerufen werden, um seine Arbeit zu verrichten. Die gesamte Kommunikation mit dem Netzwerk wird über eine einzige Callback-Routine erledigt. Das Aufrufen des Stacks wird indirekt durch die Anwendung ausgeführt. Zur Verarbeitung von Pull-Quellen sind zwei Mechanismen auswählbar(Siehe S.59). Die synchrone Verarbeitung von Pull-Quellen verhindert mögliche Seiteneffekte mit einem unterbrochenen Anwendungsprogramm, wenn die zur Abholung von Messdaten bei der Bibliothek registrierte Callback-Routine ausgeführt wird. Dieser Mechanismus ist mit der Anwendung synchronisiert, indem er dem Stack-Aufruf vorangestellt wird. Die Anwendung muss weiterhin berücksichtigen, welche Datenraten sie erwartet, um die Bibliothek (und letztendlich den Stack) oft genug aufzurufen.

6.3.1 Der Protokolladapter

Der Protokolladapter wurde im Modul `umace_protocol` implementiert. Der Protokolladapter wickelt bereits den Versand von Nachrichten mit dem in in Kapitel 5.3.7 beschriebenen Verfahren ab. Das Verfahren ist wegen der Grundstruktur einer Ringstruktur auf eine beliebig große Ringqueue erweitert worden. Für den Betrieb Bibliothek wird weiterhin der Einsatz von zwei Puffern empfohlen, doch wenn es den Bedarf geben sollte, mit einem insgesamt größeren Puffer zu experimentieren, kann die `#define`-Anweisung `UMACE_TX_BUFFER_COUNT` in der Datei `umaced.h` beliebig hoch gesetzt werden. Hierdurch könnten zum Beispiel auch exotischere Scheduling-Strategien unterstützt werden. Der maximale Datendurchsatz wird hierdurch natürlich nicht positiv beeinflusst. Die Verfügbarkeit von Pufferspeicher könnte helfen, Zeitliche Abschnitte zu überbrücken, in denen der Stack nicht aufgerufen wird. Die zum Versenden der Daten notwendige Sendezeit muß selbstverständlich wieder „aufgeholt“ werden. In einem losen System wie einer Superloop-Architektur, welche größere Schwankungen in der Periodendauer aufweist, können zunächst Testreihen durchgeführt werden, in denen alle Anwendungsfälle, die zur Beeinflussung der Korrelation der Periodendauer führen können, kurz durchgespielt werden. Der Füllstand des gesamten Puffers müsste hierbei durch die Anwendung auswertbar sein. Im Prinzip geht dies jetzt schon sehr einfach. Die verwendete Puffer-Datenstruktur enthält alle notwendigen Informationen für eine Erhebung dieser Daten. Das Eventsystem benachrichtigt letztendlich allerdings auch die Anwendung, falls der Sendepuffer übergelaufen ist und Nachrichten verworfen wurden.

6.3.2 Die Zeitstempeluhr

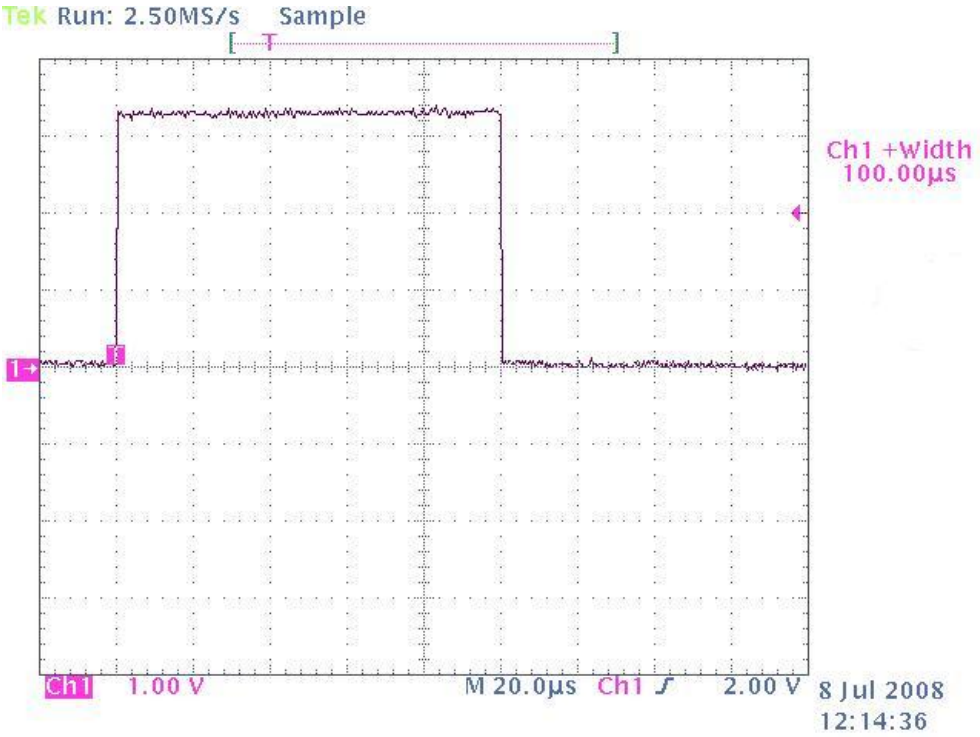


Abbildung 6.7: Messung des Timers mit einem Tektronix 544A

Die Zeitstempeluhr wird mit einer Periode von 100 μs betrieben. Der Takt der CPU wurde über die PLL auf exakt 70MHz eingestellt. Diese Einstellungen werden in `startup.s` vorgenommen. Die Zeitstempel-Uhr des Systems wird mit Timer 2 betrieben. Der Timer bekommt `PCLK=CCLK` = 70 MHz, welche danach durch den Prescaler auf exakt 10MHz heruntergeteilt werden. Das Match-Register ist auf 1000 eingestellt, was im reset-on-match Modus eine Periode von exakt 100μs ergibt. Der Vorteil dieser Auflösung ist, dass die Zeitstempelauflösung im Timer-Counter-Register auf bis zu 1μs aufgelöst werden kann. Die Zeitstempel-Periode ist zwar auf 100 μs eingetellt, doch die Uhr kann die Zeit dennoch feiner auflösen. Pull-Quellen werden durch die Zeitstempel-Uhr angetrieben. Hier sind keine Vorteile zu erkennen. Push-Quellen können allerdings jederzeit durch die Anwendung bedient werden. Hier führt dieser Ansatz zu einer deutlichen Präzisionssteigerung.

Frequenz F = 70 MHz
Periodendauer t = $\frac{1}{F} = \frac{1}{70*10^6 Hz} = 14ns$
Dauer einer AD-Wandlung auf dem LPC2468: $2,5\mu s$
$1\mu s \hat{=}$ 70 Taktzyklen $\approx$ 36 Instruktionen
$10\mu s \hat{=}$ 700 Taktzyklen $\approx$ 360 Instruktionen
$100\mu s \hat{=}$ 7000 Taktzyklen $\approx$ 3600 Instruktionen
Kleinste Abtastrate: $100\mu s \doteq 10KHz$
Kleinste Signalfrequenz: $\leq 5KHz$ (nach Shannon)

6.3.3 Der Pull-Mechanismus

Der Pull-Mechanismus wurde bisher nur im Synchronen Modus getestet. In diesem Modus wird durch die Zeitstempel-Uhr innerhalb der ISR pro registrierter Pull-Quelle ein Counter geführt. Die Pull-Perioden werden eigenen Datenstrukturen verwaltet, da die Datenstrukturen die das Modell beschreiben, nicht im Arbeitsspeicher liegen. Die Zeitstempel-Uhr setzt ein Flag um dem Synchronen Pull-Mechanismis zu signalisieren, dass es etwas zu messen gibt. Die Abtastperioden der Quellen müssen ein ganzzahliges Vielfaches der eingestellten Zeitstempelauflösung sein.

$$t_p \stackrel{!}{=} n \cdot t_c \tag{6.2}$$

t_p Abtastperiode
t_c Zeitstemel-Periode
$n \in \mathbb{N}, n \neq 0$

6.3.4 Sequenzdiagramm der C-Bibliothek

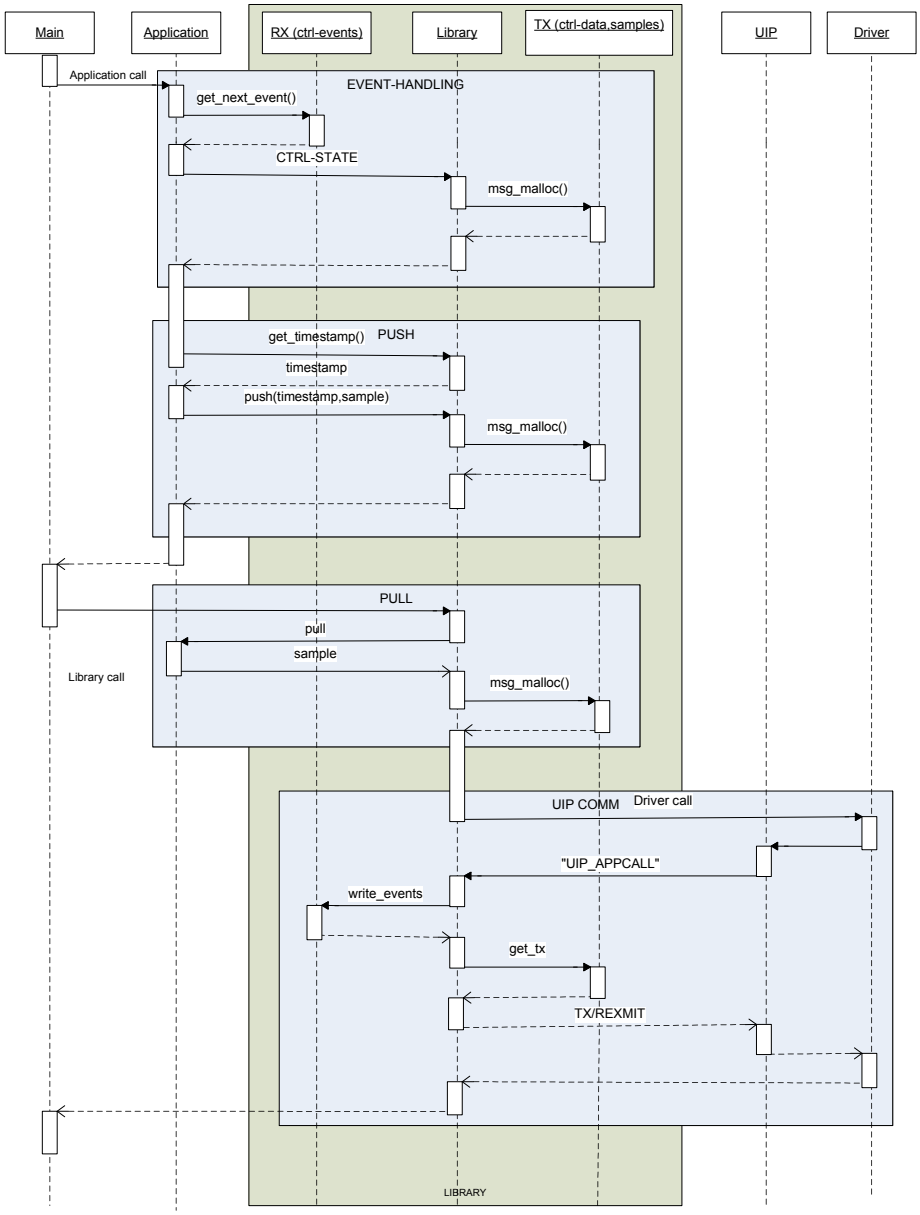


Abbildung 6.8: Sequenzdiagramm

Abbildung 6.8 zeigt die einen möglichen Ablauf der wichtigsten beteiligten Komponenten des Systems. Der Ablauf beschreibt einen Anwendungsfall in dem die Bibliothek bereits initialisiert ist und sich mitten in der Arbeitsphase befindet. Die Abbildung zeigt den grundsätzlich den wechselseitigen Aufruf

einer Anwendungsroutine und der Bibliotheksfunktion, die regelmäßig aufzurufen ist. Um genauer beschreiben zu können, wie der Datenfluss zwischen der Anwendung und der Bibliothek stattfindet, sind die Daten, die durch den Sende- und Empfangspuffer der Protokollverarbeitungseinheit verarbeitet werden, als eigene Lebenslinien aufgetragen. Die Nachrichten zwischen den Lebenslinien dieser beiden Elemente und den restlichen Lebenslinien sind mit den Funktionsaufrufen beschriftet, welche an der Verarbeitung dieser Nachrichten beteiligt sind. Dieses Diagramm soll sowohl einen möglichen sequenziellen Ablauf der Komponenten, als auch den Datenfluss zwischen der Bibliothek, der Anwendung und dem IP-Stack darstellen.

Die Vorgänge Diagramms einen Fall in dem die folgenden Ereignisse nacheinander geschehen:

Eventdaten

- Die Anwendung greift auf das Eventsystem zu.
- Die Anwendung hat sendet eine CTRL-STATE Nachricht über eine spezielle Funktion der Bibliothek.
- Die Bibliothek reserviert daraufhin Speicher im nächsten Nachritenpaket der Protokollverarbeitungseinheit

PUSH

- Die Anwendung besorgt sich einen Zeitstempel aus der Zeitstempel-Uhr
- Die Anwendung ruft die Push-Routine der Bibliothek auf.
- Die Bibliothek reserviert daraufhin Speicher im nächsten Nachritenpaket der Protokollverarbeitungseinheit

Bibliotheksaufruf (Library Call)

- Die Bibliothek wird aufgerufen. Die Bibliothek ist für die synchrone Behandlung von Push-Quellen konfiguriert. Die Zeitstempel-Uhr hat signalisiert, dass es Pull-Quellen abzuholen gibt.
- Die Bibliothek ruft den registrierten Funktionspointer der Anwendung auf und holt den Messwert.
- Die Bibliothek reserviert daraufhin Speicher im nächsten Nachritenpaket der Protokollverarbeitungseinheit

Stackaufruf durch Bibliothek

- Bevor uIP arbeiten kann, muss der Treiber aufgerufen werden. Der Treiber kopiert die eingegangenen Pakete aus dem Speicher des Ethernetcontrollers und übergibt die Kontrolle an uIP.
- uIP ruft die als Callback-Funktion realisierte Socket-Schnittstelle auf und übergibt das Paket der Protokollverarbeitungseinheit.
- Die Protokollverarbeitungseinheit schreibt ein Event in das Eventsystem
- Die Protokollverarbeitungseinheit hat festgestellt, das ein Nachrichtenpaket versendet werden muss (entweder ist die Zeit abgelaufen oder das Paket war voll)
- Die Protokollverarbeitungseinheit prüft, ob ein Paket neu zu übertragen ist (uIP überlässt dies der Anwendung)
- Die Protokollverarbeitungseinheit übergibt das Paket an uIP.
- uIP übergibt das Paket an den Treiber und überläßt dem Treiber die Kontrolle.
- Der Treiber übergibt das Paket dem Netzwerkcontroller

Es läßt sich sehr gut erkennen, welche Komponenten in einer Superloop-Architektur über den Kontrollfuss verfügen und in die Berechnung des Worst Case mit einbezogen werden müssen. Der Worst Case ist der Fall, in dem alle beteiligten Komponenten die längste Zeit gearbeitet haben und dazu geführt haben, das verhältnismäßig spät aufgerufen wurde.

6.4 Berechnungen zum Betrieb

Durch die Kenntnis der konkreten Formate der Nachrichten, können an dieser Stelle einige grundsätzliche Berechnungen durchgeführt werden. Wie in Kapitel 3.3.2 beschrieben berechnet sich die erwartete Datenrate wie folgt:

$$p = \frac{s}{t_r + t_d + t_a} \tag{6.3}$$

p Maximaler Datendurchsatz
s Segmentgröße in Bytes
t_r Round-Trip-Time RTT
t_d Delayed Acknowledge Timeout
t_a Maximale Zeit, die eine Anwendung in einem Aufruf benötigt

Auf einem System mit mit einer erwarteten Round-Trip-Time von 2 ms ergibt sich eine theoretische maximale Datenrate von 730 KB/s, was einer Bitrate von 5,8Mbit/s entspricht. Dies ist bereits die Nutzlast, der Overhead von 40 Kilobyte für den Header des Ethernet-Frames von maximal 1500 Bytes wurde bereits abgezogen.

6.4.1 Berechnung der Häufigkeit der Bibliotheksaufrufe

Bei der Dimensionierung eines Mess- und Regelsystems, welches Gebrauch von dieser Bibliothek macht, ist es wichtig, vorher zu wissen, welches Datenaufkommen zu erwarten ist. Dies ist notwendig, da im Scheduling-Verfahren der Anwendung der Stack der Bibliothek aufgerufen werden muss, um die anfallenden Daten zu versenden. Grundsätzlich sollte hierfür die Häufigkeit aller ausgehenden Nachrichten im System abgeschätzt, beziehungsweise möglichst genau bestimmt werden. Zu diesen Nachrichten zählen vor allem die SAMPLE Nachrichten, da diese Nachrichten das meiste Datenvolumen erzeugen sollten. Die CONTROL-STATE Nachrichten sollten nicht annähernd so viele Daten produzieren.

Typische Konfigurationen

Es folgen ein paar Rechenbeispiele für mögliche typische Konfigurationen: Jede ausgehende SAMPLE Nachricht hat einen Overhead von 12 Bytes. (msgid+length+timestamp+srcid), Bei einer Samplegröße von 1 Byte entstehen also 13 Bytes im Sendepuffer der Bibliothek.

256 Quellen a 1Hz, Samplegröße=1Byte
$13Bytes \widehat{=} 104Bit$
$256 \cdot 104Bit \cdot 1Hz \approx 0.027Mbit/s$

256 Quellen a 1Hz, Samplegröße=2Byte
$256 \cdot 112Bit \cdot 1Hz \approx 0.029Mbit/s$

1 Quelle a 1KHz, 10 Quellen a 1Hz, Samplegröße=1Byte $10 \cdot 104Bit \cdot 1Hz +$
$1 \cdot 104Bit \cdot 1KHz = (1,04 + 104)KBit/s \approx 0.1MBit/s$

10 Quellen a 1 KHz, Samplegröße=1Byte
$10 \cdot 104Bit \cdot 1KHz = 1,04MBit/s$

Theoretische Maxima

1 Quelle, Samplegröße=1Byte,Maximale Samplerate bei einer Sendeleistung von 5 MBit/s
CPU: 1.9 CPI, 72MHz, $1\mu s \widehat{=}$ 72 Cpu-Takte ≈ 37 *Instruktionen*

$$\frac{5MBit/s}{104Bit} \approx \frac{50000}{s} = 50\ KHz \rightarrow Periodendauer\ 20\mu s \tag{6.4}$$

Hier müsste die Zeitstempeluhr auf $20\mu s$ eingestellt werden, damit ihre zeitliche Auflösung dies überhaupt auflöst. Bei einer Theoretischen maximalen Abtastrate von 50 KHz muß die Zeitstempel-Uhr mindestens 50 KHz betrieben werden. Hierbei muß berücksichtigt werden, das die Cpu genügend „Reserven“ hat.

$$20\mu s \cdot \frac{37\ Instruktionen}{1\mu s} \widehat{=} 740\ Instruktionen \rightarrow\ etwasKnapp.. \tag{6.5}$$

Jetzt betrachten wir einmal, wie viele Nachrichten in diesem Fall pro Paket versendet werden. Bei dem verwendeten Sendeverfahren wird auf den nächsten Sendepuffer umgeschaltet, falls eine Nachricht nicht mehr in den Puffer passt(siehe Abbildung 5.23 auf Seite 67). Der verbleibende Rest bleibt ungenutzt. Hierdurch kann nicht immer die maximale Datenrate ausgenutzt werden. Sendepuffer=1460 Bytes Paketgröße=13 Bytes

$$\frac{1460Bytes}{13Bytes} = 112,3 \tag{6.6}$$

Es passen 112,3 Samples in den Sendepuffer. Abgerundet ergibt das 112 Samples. Die maximale Sendeleistung wäre in diesem Fall also eigentlich nur (112*13/1460) * 5MBit gewesen, also 99,7% davon.

Den größten Einfluss haben Nachrichtengrößen, die sich um den Bereich (1460/2+1) bewegen.

$$\frac{1460Bytes}{731Bytes} \approx 1,99 \implies 50\% \tag{6.7}$$

Abbildung 6.9 zeigt die Auswirkungen dieses Zusammenhangs. Die Formel hierzu lautet:

$$f(x) = \frac{floor(\frac{1460}{x}) \cdot x}{1460} \tag{6.8}$$

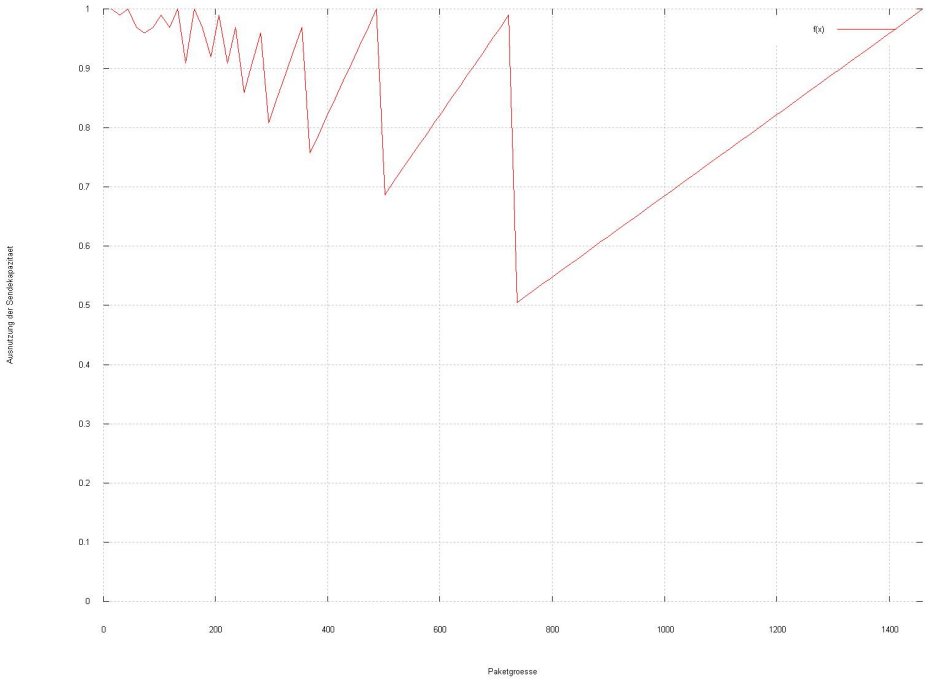


Abbildung 6.9: Ausnutzung der Sendekapazität des Sendeverfahrens

Grundsätzlich wird die Häufigkeit der Bibliotheksaufrufe abgeschätzt, in dem die Datenraten abgeschätzt werden, die maximal zu zu erwarten sind. Handelt es sich um kleine Nachrichten bis 200 Byte darf von 90% der maximal gemessenen Durchsatzrate ausgegangen werden. Bei Paketen bis 400 Byte darf nur noch von 75% der maximalen Datenrate ausgegangen werden (Siehe Abbildung 6.9).

Bei der Verwendung von SAMPLE-Nachrichten von 1-4 Byte ist dieser Effekt absolut unkritisch.

$$t_b = \frac{P_{aw} \cdot X}{1460} \tag{6.9}$$

P_{aw} Max. Datenrate der Anwendung
X Sicherheitsfaktor
t_b Periode des Bibliotheksaufrufs

Systemen die sich an den Extremen bewegen, verbleibt wenig Rechenzeit für die Anwendung. Grundsätzlich sollte vorher die benötigte Rechenzeit der Anwendung ermittelt werden, um diese in die Berechnung für den maximalen Datendurchsatz einzubeziehen (Siehe 6.1).

6.4.2 Optimierungen

- Durch das Hinzufügen eines Periodenoffsets für Pull-Messquellen kann die auftretende Last des Systems besser über die Zeit verteilt werden.
- Eine Priorisierung der Messquellen könnte helfen, im Pull-Modus die unwichtigsten Quellen vorrübergehend zu deaktivieren, um den wichtigen Quellen Vorrang zu lassen, falls beispielsweise Leitungsstörungen temporär für höhere Round-Trip-Zeiten sorgen.

6.5 Aufbau des Java-Applets

Das Datenmodell des Java-Applet basiert auf den in der Katalogphase (Siehe Seite 73) des Protokolls empfangenen Daten. Diese Daten bilden die Grundlage für die Verarbeitung des Protokolls. Das Gegenstück zur Protokollverarbeitungseinheit der C-Seite bilden die Klassen `Connector` und `ProtocolAdapter`. Die Klasse `Connector` wickelt die Kommunikation mit dem Socket ab(Hier wird das *Delayed Acknowledge* ausgeschaltet) und die Klasse `ProtocolAdapter` ist für die Implementierung des Protokolls zuständig. Die Klasse `Catalog` dient als zentraler Speicherort für die Klassen `Source`,`Context` und `Control`, welche das Datenmodell abstrahieren. Der Protokolladapter instanziert eintreffende Katalogelemente und legt übergibt sie an die Klasse `Catalog`. Beim Instanzieren der Klasse `Source` wird über die Klasse `Classfinder` der unter dem Typ-Feld des Source-Datensatzes gespeicherte Typ-String verwendet, um eine zugehörige Darstellungsklasse zu finden. Die Darstellungsklassen auf Java-Seite müssen von der abstrakten Klasse `SourceDisplay` abgeleitet sein. Analog dazu geschieht die Instanzierung einer Klasse für ein Bedienelement, welche das Interface `ControlInterface` implementieren muss. Die Darstellungsklassen für Kontext-Typen müssen von JPanel abgeleitet sein. Bei der Instanzierung bekommen diese Klassen die aus den empfangenen Katalogdaten durch die Helfer-Klasse `Converter` erzeugten Properties-Objekte.

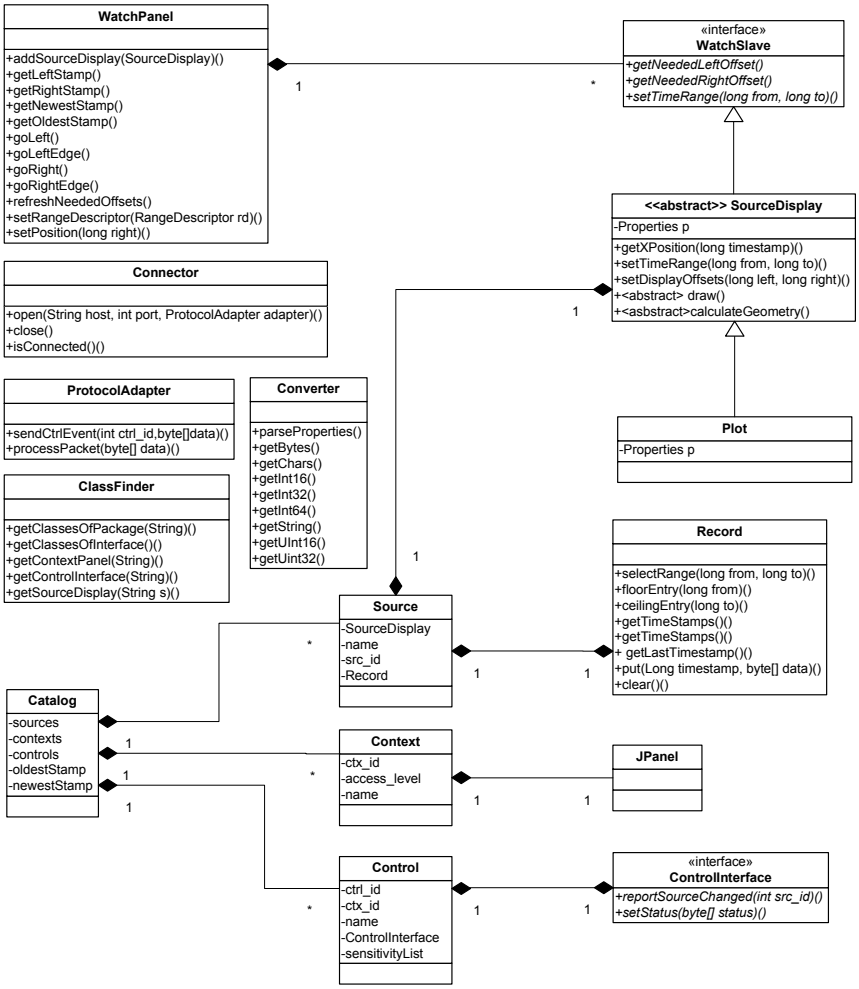


Abbildung 6.10: UML-Diagramm der Software

Die Klasse `Catalog` dient als Dispatcher, wenn neue SAMPLE oder CONTROL-STATE - Nachrichten eintreffen. Diese Nachrichten werden direkt an die Instanzen der Typ-Klassen(`SourceDisplay` und `ControlInterface` weitergeleitet. SAMPLE-Nachrichten werden in der Klasse `Record` abgelegt, welche Instanzen der Klasse `SourceDisplay` als Datenspeicher dient. Die Klasse `Plot` realisiert eine die Funktionalität zum Zeichnen einfacher Funktionsplots.

Durch die Benutzeroberfläche können Messquellen in der Klasse `WatchPanel` dargestellt werden. Das Interface `WatchSlave` regelt die Synchronisation mit den aktivierten Messquellen-Displays.

6.5.1 Entwicklungsstand

Das Java-Applet ist weit fortgeschritten. Was noch fehlt, ist das Zeichnen der Zeitleiste und die Skalen. Alle auf den Folgenden Bildern enthaltenen Displayelemente sind zur Verfügung stehende Typen.

6.6 Bilder der Oberfläche

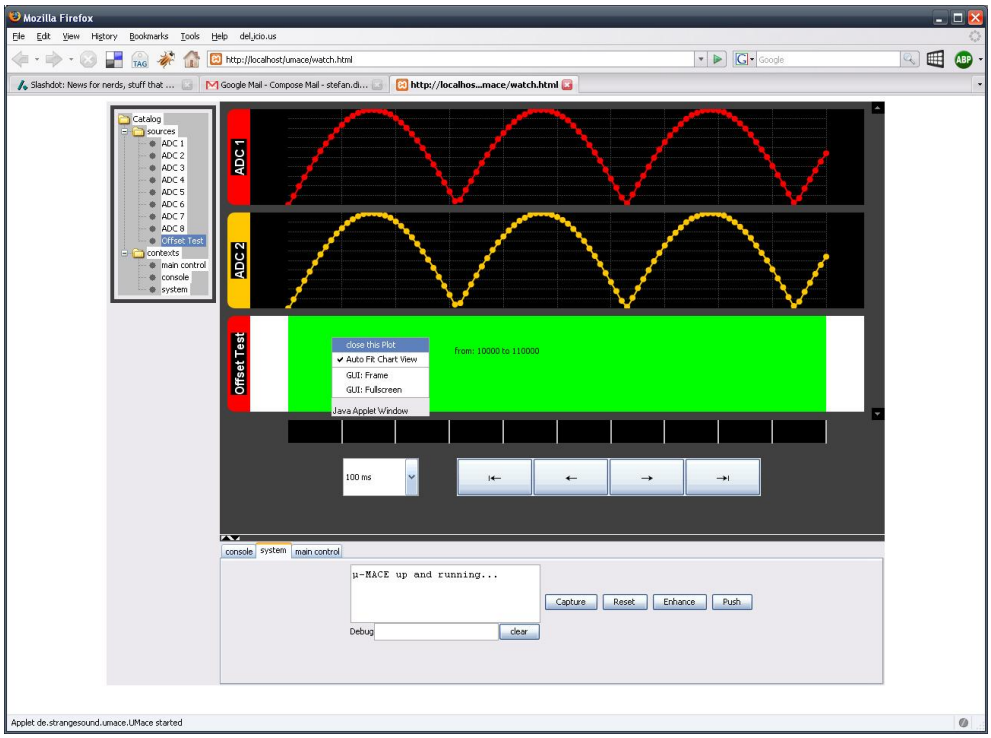


Abbildung 6.11: μ-mace im Browser mit Control-Bereich und SidePanel

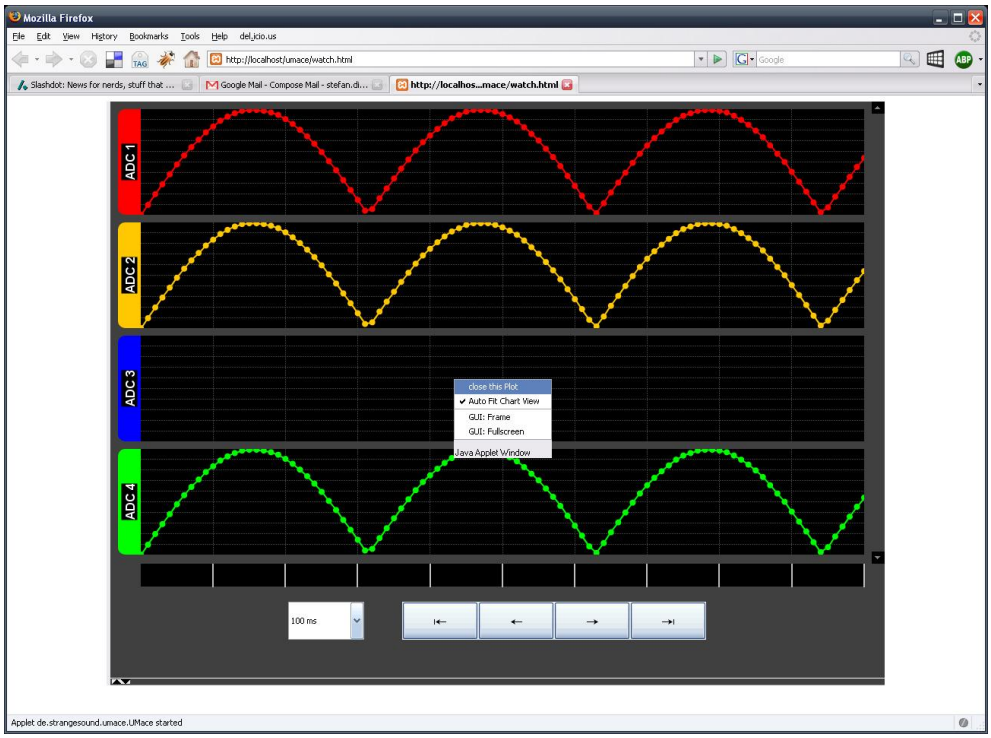


Abbildung 6.12: μ-mace im Browser - Panels ausgeblendet

Das Package umace.event enthält die Klasse `ContextMenuSpawner` und zwei Annotationen. Annotationen sind Meta-Informationen, mit denen in diesem Fall durch den Programmierer Methoden markiert werden können. Die Annotation `@Command(accessLevel=0, name="close this Plot")`. markiert eine Methode, welche den zum Schließen benötigten Befehl enthält. Alle erweiterbaren Klassen können über diese Fähigkeit verfügen. Damit Displaybereiche schön aufgeräumt bleiben, kann viel Funktionalität auf die Maus verlagert werden. Es gibt auch eine Annotation für einen Checkbox-Eintrag auf der rechten Maustaste. Komponenten können über diesen Mechanismus Funktionen an- und ausschalten.

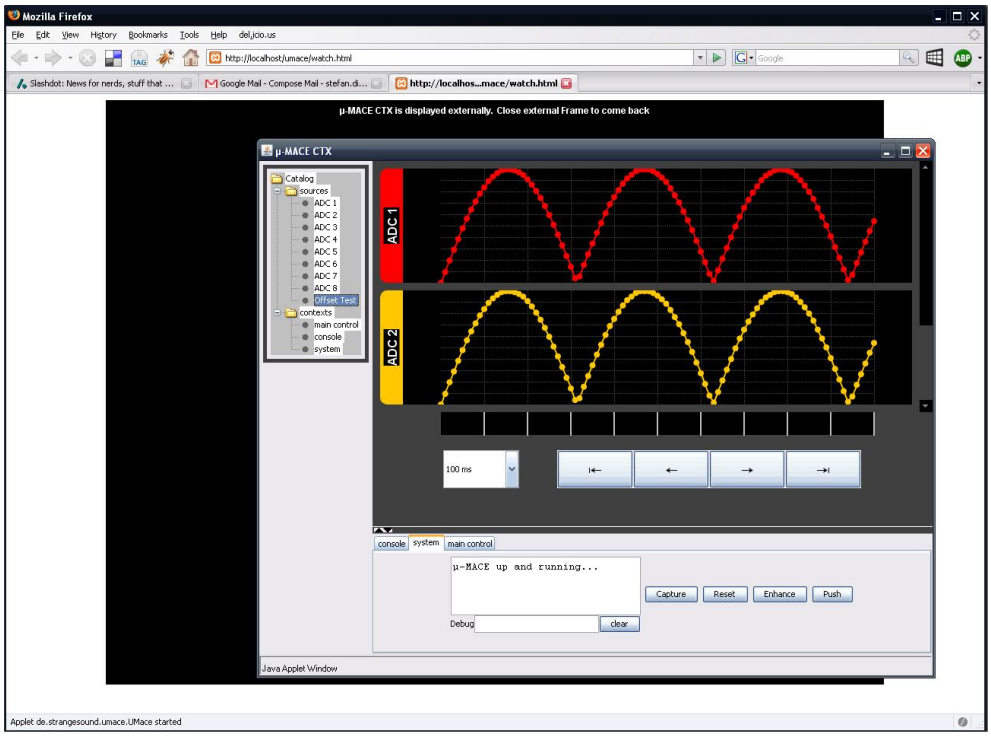


Abbildung 6.13: μ-mace im eigenen Frame

Das Programm lässt sich auch aus dem Browser herauslösen.

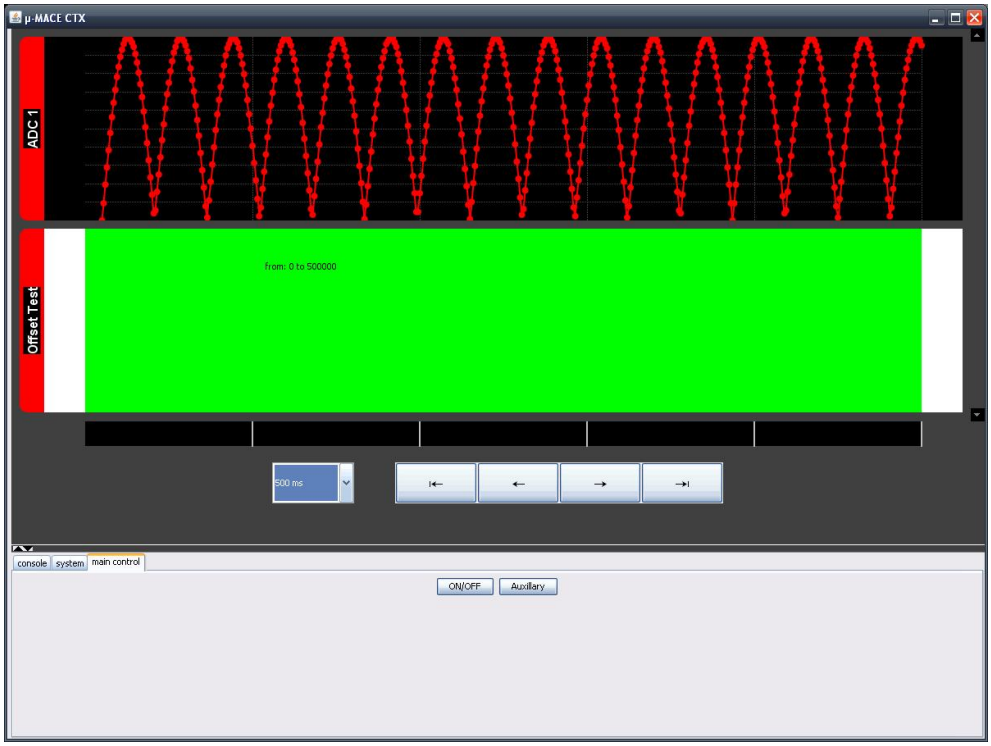


Abbildung 6.14: Eine SourceDisplay-Klasse reserviert Displayfläche

```
package de.strangesound.umace.types.sources;

import java.awt.Color;

public class OffDisplay extends SourceDisplay {

  public OffDisplay(Source s)
  {
    super(s);
  }

  @Override public void draw(Graphics2D g2d)
  {
    Rectangle2D rect = new Rectangle2D.Float(graphStartX, 0, graphWidth, panelHeight);
    Rectangle2D plotrect = new Rectangle2D.Float(timeStartX,0,timeWidth,panelHeight);
    g2d.setColor(Color.WHITE);
    g2d.fill(rect);
    g2d.setColor(Color.GREEN);
    g2d.fill(plotrect);

    // liefert eine Map<long,byte[]> mit den Werten des Zeitlichen Abschnitts:
    //source.getRecord().selectRange(startStamp, endStamp);

    g2d.setColor(Color.BLACK);
    g2d.drawString("from: " + startStamp + " to " + endStamp ,300,50 );
  }
  @Override public void calculateGeometry(){}

  @Override public int getNeededLeftOffset(){ return 50;}

  @Override public int getNeededRightOffset(){return 50;}

}
```

Abbildung 6.15: Der Source-Code für ein SourceDisplay

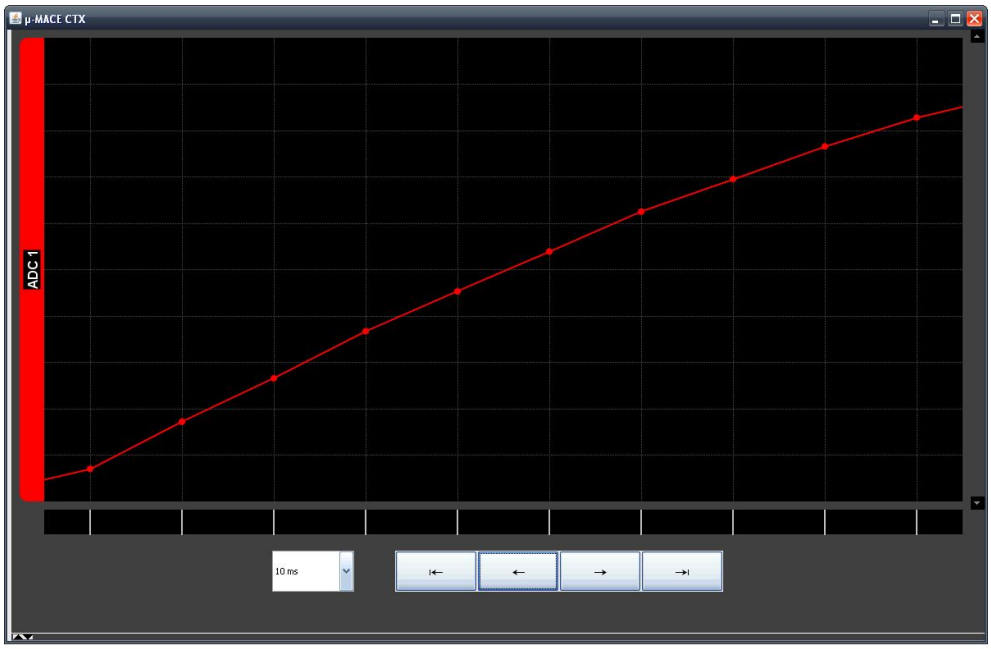


Abbildung 6.16: Eine Quelle, kleine Zeitbasis

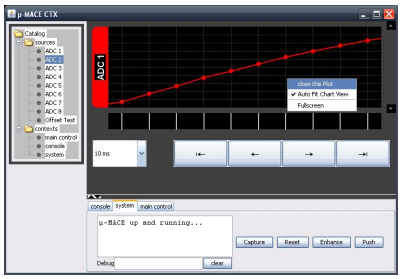


Abbildung 6.17: μ-mace ganz klein

Auch in dieser Größe kann man noch damit Arbeiten. Die Sidebar ist gerade „festgeklebt“. Ein rechter Mausklick und sie verschwindet an die linke Bildschirmkante. Zwischen den Quellen kann per Mausrad umgeschaltet werden. Die Displaykomponente, welche die SourceDisplays beinhaltet, ist komplett in Java2D entstanden. Sobald das Mausrad bewegt wird, schaltet der Anzeigebereich eine Quelle weiter, beziehungsweise zurück. Die Konsole im unteren Displaybereich ist eine Bedienkomponente, welche durch den Mikrocontroller registriert werden kann. Die CONTROL-EVENT und CONTROL-STATUS - Nachrichten sind einfach nur Strings.

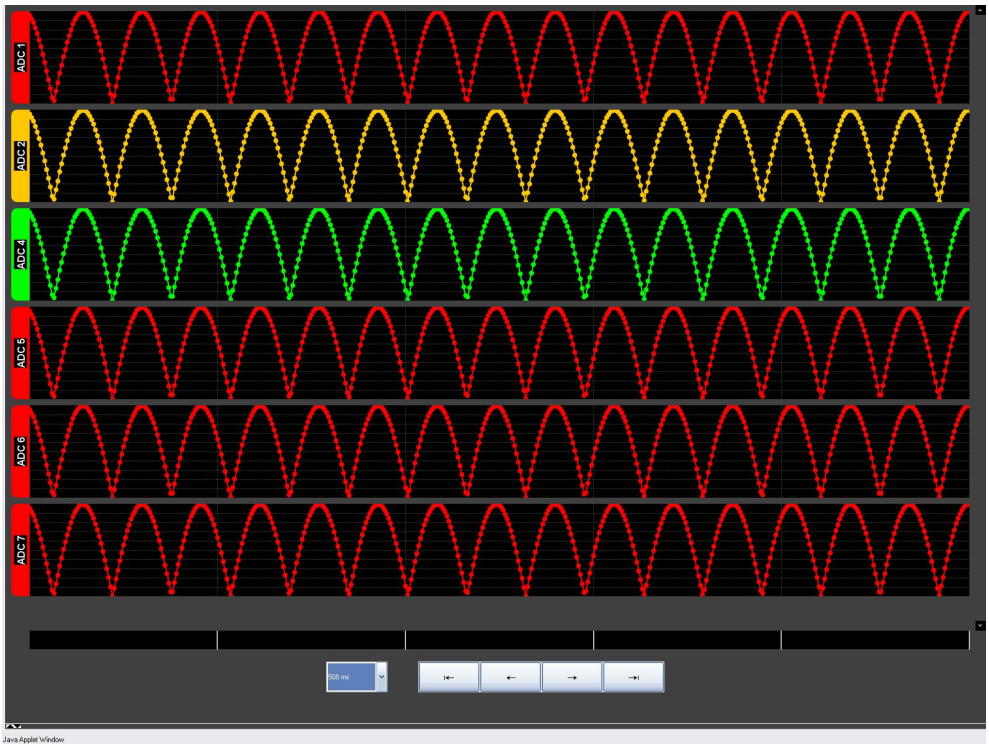


Abbildung 6.18: μ-mace fullscreen

Das Programm wurde ursprünglich aus dem Browser heraus gestartet. Jetzt ist das Programm vollständig maximiert.

Abbildung 6.19: μ-mace CTX -Micro Measurement and Control Environment

7 Zusammenfassung und Ausblick

Die entstandene Bibliothek stellt keine hohen Anfordeungen an das Scheduling einer Anwendung. Die Aufgaben der Bibliothek werden durch den regelmäßigen Aufruf einer Bibliotheksfunktion erledigt, dessen Häufigkeit je nach benötigter Datenrate bestimmt werden kann. Der Einsatz eines eigenen Timers ist bei dem gezeigten Konzept notwendig, doch die Software ist leicht an die Bedürfnisse der eigenen Umgebung anpassbar.

Der Benutzer kann bei der Auswertung der Daten im Browser nicht nur die Live-Daten betrachten, er kann auch den zeitlichen Ausschnitt sowie eine Zeitbasis verändern, um die Veränderung der Daten über die Zeit auszuwerten. Die Einbindung der durch die Bibliothek bereitgestellten Mechanismen könnte einfacher nicht sein. Das enthaltene Java Applet wird als leere Hülle an den Browser ausgeliefert und nimmt nach Aufruf der Webseite Kontakt zur C-Seite der Bibliothek auf. Auf Mikroprozessorseite wird durch den Programmierer über einheitliche Deskriptortabellen ein Beschreibungsschema für die benötigten UI-Komponenten der Anwendung bereitgestellt, mit welchem das Applet initialisiert wird. Die Oberflächenkomponenten im Browser werden der Anwendung auf dem Mikroprozessor über ein Eventsystem verfügbar gemacht. Die Messdaten der Anwendung werden bei der Bibliothek über eine einheitliche Schnittstelle registriert und können sofort in einer Art grafischem Datalogger betrachtet werden. Das Java-Applet verwendet den Speicher der Client-Plattform, um ein Log der anfallenden Messdaten aufzuzeichnen. Die Verlagerung aller Bedienkomponenten in die Browseroberfläche bietet die Möglichkeit, die Portabilität der Anwendung zu erhöhen, da keine systemspezifischen Hardwareschnittstellen angesprochen werden müssen, um beispielsweise Displays und sonstige Hardware-Benutzerschnittstellen anzusprechen. Die verwendeten Komponenten werden darüber hinaus durch das Eventsystem über einen einheitlichen Mechanismus an zentraler Stelle verfügbar gemacht. In der Entwicklungsphase einer Anwendung kann die Bibliothek sehr schnell dazu dienen, Schnittstellen zur Benutzerinteraktion bereitzustellen, wie zum Beispiel ein simples Terminal im Browser, über das der Programmierer Ein- und Ausgaben machen kann. Knöpfe, Schieberegler, sonstige UI-Komponenten - alles steht im Handumdrehen zur Verfügung.

A Der verwendete Prozessor

Geschichte

Die *ARM*[1]-Architektur ist eine weit verbreitete 32-Bit Risc-Prozessorarchitektur, die in vielen Embedded-Designs Einzug gefunden hat. Die ARM Prozessoren finden sich heutzutage größtenteils in mobilen Geräten, wo niedriger Energieverbrauch zu den wichtigsten Designkriterien zählt. Das ARM-Design wurde 1983 als Entwicklungsprojekt bei *Acorn Computers Limited* gestartet. Im Jahre 1986 wurde das Design des ARM2 fertiggestellt, welches damals mit nur 30.000 Transistoren (im Vergleich zu Motorolas 6 Jahre älterem Prozessormodell *68000* mit ca 70.000 Transistoren) als das wahrscheinlich einfachste 32-Bit Prozessordesign galt. Laut [27] hat Acorn damals das weltweit erste kommerzielle single-chip RISC-Prozessordesign entworfen. Vergleichbare Architekturen waren zu diesem Zeitpunkt nur in High-End-Workstations, die auf maximale Performanz abzielten, zu finden. In den späten Achztigern begann Acorn zusammen mit *Apple Computers Ltd.* an neueren Versionen des Core-Designs zu arbeiten. Die Arbeit an dem neuen Prozessordesign stellte sich im November 1990 als so wichtig heraus, dass das Designteam als Joint Venture zwischen *Apple Compuers Ltd.*, der *Acorn Conputer Group* und *VLSI Technology* in das neue Unternehmen *Advanced Risc Machines Ltd.* abgespaltet wurde. Diese Arbeit führte zum ARM6-Design, dem ersten embedded-RISC Prozessor. Die Größe des Prozessorkerns wuchs hierbei auf nur ca. 35.000 Transistoren an. Die ersten Modelle wurden 1991 in Form des ARM610 für den *Apple Newton* PDA ausgeliefert. Acorn benutzte den ARM610 1994 als Basis für den *Acorn Risc PC 600*. Advanced RISC Machines wurde zu *ARM Limited* als das Unternehmen im April 1998 am die Börse ging. Laut[27] ist ARM im Jahre 2005 ein globales Unternehmen mit mehr als 1.250 Angestellten und Zweigstellen in 12 Ländern.

IP-Core

Das Geschäft von ARM beschränkt sich auf den Verkauf von *IP Cores*[2]. Dies bedeutet, dass lediglich das Prozessordesign an Lizenznehmer verkauft wird. ARM selbst stellt keine Hardware her. Die Lizenznehmer verwenden das Design, um eigene Mikrocontroller auf Basis dieses Designs zu entwickeln. ARM bietet eine Vielzahl an maßgeschneiderten Lizenzen an. Es gibt Lizenzen zur

[1] Advanced Risc Machine, davor Acorn Risc Machine
[2] Intellectual Property Core

Erweiterung und Herstellung des Designs in Form eines eigenen Produkts. ARM liefert sowohl die Hardwarebeschreibung des ARM-Prozessorkerns, als auch die komplette Software, um dieses Design zu testen und zu erweitern. Lizenznehmer, die das ARM Design in ihre eigenen Chipdesigns integriern möchten, sind normalerweise nur daran interessiert, einen für die Herstellung fertig verifizierten Prozessor zu erwerben. Für diese Kunden liefert ARM eine Netzliste des gewählten ARM-Designs mit einem abstrahierten Simulationsmodell des Kerns und Testprogrammen um die Designintegration und Verifikation zu unterstützen. Ambitioniertere Kunden können den Prozessor auch in synthetisierbarer *Verilog*-Form kaufen. Mit diesen Daten hat der Kunde die Möglichkeit, architekturelle Optimierungen und Erweiterungen vorzunehmen. Dies ermöglicht den Lizennehmern, kompliziertere Modifikationen wie zum Beispiel die Erweiterung des Befehlssatzes zu erreichen, welche mit einer Netzliste nicht möglich wären. Am 22. Januar 2008 meldet[3] ARM, dass die Anzahl der durch Lizenznehmer verkauften Prozessoren die 10-*Billionen*[4]-Marke erreicht hat.

Die ARM-Architektur

Die ARM-Architektur ist eine RISC[5]-Architektur, welche typische RISC-Merkmale aufweist. Hierzu gehören ein großer Registersatz und eine Load/Store-Architektur. Datenverarbeitende Instruktionen können nur auf Registern arbeiten und nicht auf direkt adressiertem Speicher. Die ARM-Architektur verfügt über einfache Adressierungsmodi in denen alle Load/Store Adressen nur aus Registerinhalten und Instruktionsfeldern entnommen werden. Die Instruktionsfelder im Instruktionscode sind alle gleich lang, um die Instruktionsdekodierung zu vereinfachen. Es können mehrere Instruktionen in einem Load-Zyklus geladen werden, um den Datendurchsatz zu maximieren. Ein besonderes Feature der ARM-Architektur ist die bedingungsgebundene Ausführung von fast allen Befehlen, sowie die zusätzliche Verarbeitung des zweiten Operranden durch einen *Barrel-Shifter*[6] bei der Ausführung von fast jeder datenverarbeitenden Instruktion. Dies führt zu einer deutlichen Reduktion des entstehenden Maschinencodes und der benötigten Ausführungszeit. Die ARM-Architektur hat im Laufe der Zeit eine Menge Veränderungen erfahren. Alle folgenden Informationen beziehen sich im Speziellen auf die technischen Eigenschaften der ARM7TDMI-S - Serie. Der ARM7TDMI-S hat eine von-Neumann Architektur mit einem einzigen Datenbus für Instruktionen und Daten. Nur `load`, `store` und `swap`-Instruktionen können auf den Speicher zugreifen.

[3] "`http://www.arm.com/news/19720.html`"
[4] wobei nicht sicher ist, ob hiermit 10^9 oder 10^{12} gemeint ist
[5] Reduced Instruction Set Computer
[6] Ein Barrel-Shifter

Der Instruktionssatz

Der ARM-Instruktionssatz stellt einen leistungsfähigen Satz von Operationen und Adressierungsmodi zur Verfügung, die einem Programmierer erlauben, kurzen, schnellen und effizienten Code zu schreiben. Neuere ARM-Versionen sind grundsätzlich abwärtskompatibel zu Code, der für ältere Versionen geschrieben wurde. Eine Ausnahme ist allerdings der ältere 26-Bit Betriebsmodus, welcher in einigen Versionen des Kerns nicht mehr unterstützt wird. Die bedingungsgebundene Ausführung von ARM-Instruktionen kann zu einer deutlichen Beschleunigung führen. Anstatt eine Vergleichsopertaion auszuführen und dann in Abhängigkeit vom Ergebnis einen bedingten Sprung auszuführen können diese *Conditionals* in-line ausgeführt werden. Mittels des `swp`-Befehls können atomare Read-and-Write Zugriffe auf den Speicher ausgeführt werden, um Semaphoren zu realisieren. Mit `ldm` und `stm` können beliebig viele Register mittels einer einzelnen Instruktion auf den Stack gesichert bzw vom Stack zurück geschrieben werden. Es gibt mehrere Instruktionen die die Anbindung von Coprozessoren an den ARM-Kern ermöglichen. Diese Instruktionen erlauben den Transfer von Speicher- und Registerinhalten von und zu etwaigen Coprozessoren sowie den Aufruf von Coprozessor-spezifischen internen Befehlen. Neben dem ARM-Instruktionssatz unterstützen neuere ARM-Prozessoren auch sogenannten *Thumb*®-Code, welcher im Gegensatz zum ARM-Instruktionssatz nur 16 Bit breit ist und durch seine Einfachheit zu etwas längerem Code führt. Zu jeder Thumb-Instruktion gibt es eine äqquivalente ARM-Instruktion. Es sind jedoch nicht alle ARM-Instruktionen auch als Thumb-Instruktionen verfügbar. Diese Instruktionen müssen im Thumb-Code durch mehrere Thumb-Instruktionen ersetzt werden. Letztendlich stellen die Thumb-Instruktionen allerdings keine eigenen Instruktionen dar, sondern werden vom Instruktionsdecoder auf ihre korresbondierenden 32-Bit ARM-Instruktionen erweitert. Der Unterschied zwischen den beiden Instruktionsarten ist, wie sie geladen werden. Da die Instruktionsdekodierung mittels dedizierter Hardware stattfindet, können diese Instruktionen genau so schnell ausgeführt werden, wie die des ARM-Instruktionssatzes. Die resultierenden Programme werden zwar etwas länger, jedoch benötigen die um die Hälfte kürzeren Instruktionen weniger Programmspeicher. Programmierer können beim Schreiben Ihrer Programme Thumb- und ARM-Instruktionscode vermischen, zum Beispiel einzelne Subroutinen mit dem jeweils anderen Instruktionssatz implementieren, um gezielt Optimierungen vorzunehmen. Interrupt-Service-Routinen müssen jedoch oft(zumindest teilweise) in ARM-Instruktionscode geschrieben werden. Der ARM-Kern muß zur Ausführung von Thumb-Code in den Thumb-Betriebszustand umgeschaltet werden. Hierzu gibt es unter anderem spezielle Branch-Befehle, die beim Subroutinen-Einsprung den Betriebszustand von ARM auf Thumb und andersherum umschalten können. Laut[22, Kapitel 1.2.2] Benötigt Thumb-Code typischerweise 65 Prozent der Größe von ARM-Code und ist 160 Prozent schneller als Dieser, wenn der Prozessor auf 16-Bit Speicher arbeitet. Prozessoren mit der *Jazelle*®-Erweiterung erlauben außerdem die native Ausführung von Java-Code auf einigen Prozessoren. Die Ausführung von Java im Jazelle-Betriebszustand ist allerdings nur mit der Un-

terstützung einer Jazelle-fähigen *JVM*[7] möglich, die für kompliziertere oder seltener benutze Bytecodes in einen Softwaremodus umschaltet. Die öffentlichen Spezifikationen sind sehr unvollständig und ermöglichen es lediglich, Betriebssysteme zu entwerfen, welche die Ausführung einer Jazelle-fähigen JVM ermöglichen. Die Dokumentation von Jazelle steht nur Entwicklern mit einer *JTEK*[8]-Lizenz zur Verfügung.

Prozessor Modi

Der Prozessor verfügt über verschiedene Betriebsmodi, mit denen sowohl die Beschleunigung von einigen Vorgängen, als auch die Abschottung von unprivilegiertem Code realisiert werden kann.

Die Modi sind:

- **User** - Normale Programmausführung
- **FIQ** - Unterstützt einen High-Speed Datentransfer Prozess
- **IRQ** - Benutzt für General-Purpose Interrupt handling
- **Supervisor** - Ein geschützter Modus für das Betriebssystem
- **Abort** - Implementiert virtuellen Speicher und/oder Speicherschutz
- **Undefined** - Unterstützt Software-Emulation von Hardware-Coprozessoren
- **System** - Führt privilegierte Betriebssystemtasks aus

Wenn sich der Prozessor im **User**-Modus befindet, ist kein Zugriff auf geschützte Speicherbereiche oder das Umschalten des Modes möglich, außer durch das Erzeugen eines Software-Interrupts, der wiederum die Realisierung eines Betriebssystems ermöglicht, welches die Verwendung von Systemressourcen verwalten kann. Alle anderen Modi werden als sogenannte *Privilegierte* Modi bezeichnet. Sie haben vollen Zugriff auf Systemressourcen und können den Mode frei umschalten. Die Betriebsmodi **FIQ**, **IRQ**, **Supervisor**, **Abort** und **Undefined** werden außerdem als Exception Modes bezeichnet. Diese Betriebsmodi werden betreten wenn die korresbondierenden Exceptions auftreten. Der **System**-Modus hat exakt die gleichen Register, wie der *User*-Modus. Es ist ein privilegierter Modus der keinen Einschränkungen unterliegt. Dieser Modus ist für die Ausführung von Betriebssystem-Tasks gedacht, die vollen Zugriff auf Systemressourcen benötigen, aber die Verwendung von zusätzlichen Registern die normalerweise mit Exception-Modes zusammenhängen vermeiden

[7] *Java Virtual Machine*

[8] Java Technology Enabling Kit, siehe `http://www.arm.com/support/faqdev/1210.html` und `http://www.arm.com/products/esd/jazelle_software.html`

möchten. Die Verwendung dieses Modus stellt sicher, dass der Taskzustand nicht zerstört wird, wenn eine Exception auftritt.

Register Shadowing

Die ARM-Architektur macht Gebrauch von sogenanntem *Register Shadowing*, dies bedeutet dass der Prozessor einige Register in Abhängigkeit vom Prozessormodus aus- und einblendet. Insbesondere die Register `R13` (Stackpointer),`R14` (Linkregister) und das Register *SPSR*[9] sind für jeden Prozessormodus gesondert ausgeführt und werden in Abhängigkeit vom Prozessormodus bankweise umgeschaltet. Abbildung A.1 zeigt eine Übersicht über die Register des ARM-Kerns in den verschiedenen Prozessormodi.

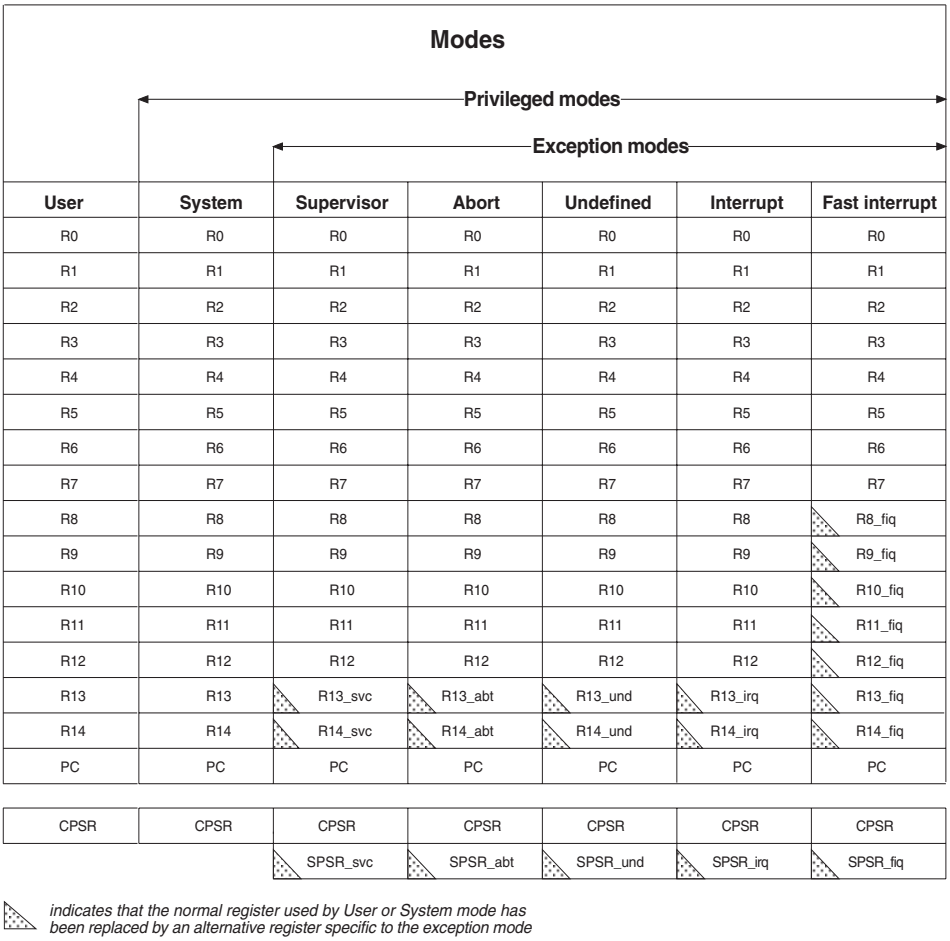


Abbildung A.1: *Register Shadowing* im ARM-Kern ©ARM Ltd.

Der ARM hat 31 Register, von denen allerdings nur die Register `R0` bis `R14` als General-Purpose-Register verwendet werden. Das Register R15 ist der Pro-

[9] *Saved Program Status Register*, auf dieses Register wird weiter unten genauer eingegangen

gram Counter(PC). Diese 16 Register sind zu jeder Zeit sichtbar, die verbleibenden Reigster werden verwendet um die Interruptverarbeitung zu beschleunigen. Die ersten 16 Register werden von unprivilegiertem Code verwendet, sie werden *User Mode Register* genannt. Im Thumb-Modus stehen anstatt der 16 Register nur die ersten acht Register zur Verfügung Die acht im Thumb-Modus verfügbaren Register sind die selben, die auch im ARM-Modus zur Verfügung stehen. Sie können somit benutzt werden, um transparent Daten zwischen den beiden Betriebsmodi auszutauschen. Im Thumb-Modus existieren eigene `push` und `pop`-Instruktionen, die im ARM-Modus nicht verfügbar sind. Diese Instrukitonen werden in `ldm`- und `stm`-Instruktionen umgesetzt. Der User-Mode unterscheidet sich von allen anderen Modi, da er unprivilegiert ist. Dies bedeutet, dass der Benutzer nur zu einem anderen Prozessormodus umschalten kann, wenn er eine „Exception“ auslöst. Der Software-Interrupt(SWI) ermöglicht diese Fähigkeit aus dem Programm heraus. Das Speichersystem und mögliche Coprozessoren können den Zugriff auf den Speicher aus dem User-Mode heraus einschränken. Drei der 16 sichtbaren Register haben spezielle Rollen. `R13` ist der Stack-Pointer und `R14` ist das Link-Register. Das Link-Register hält die Adresse der nächsten Instruktion nach einer *Branch-and-Link*[10]-Instruktion. Es wird außerdem verwendet um die Rücksprungadresse bei einem Interrupt zu speichern. Zu jeder anderen Zeit steht `R14` zur normalen Verwendung zur Verfügung. Das Register `R15` ist der Program-Counter(PC). Es hält die Adresse der als nächstes auszuführenden Instruktion. Diese drei Register werden außerdem auch im Thumb-Betriebszustand verwendet.

Eine Besondere Bedeutung kommt dem `CPSR`[11] zu. Dieses Register hält den Prozessormodus (User- oder Exception-Flag), die Interruptmasken-Bits, die Condition-Flags und das Thumb-Statusbit. Obwohl die anderen Bits im `CPSR` durch die Software verändert werden können, darf das Thumb-Statusbit nicht direkt modifiziert werden. Das Ergebnis dieser Modifikation ist nicht vorraussagbar. Jeder Modus verfügt ein eigenes `SPSR`(*Saved Program Status Register*), welches den Zustand des `CPSR` vor Auftreten der Exception speichert. Der Zugriff auf das Register `CPSR` findet normalerweise über spezielle Instruktionen statt.

Exceptions

Die ARM-Architektur unterstützt sieben Typen von Exceptions für die neben dem User-Mode, (welcher übrigens sowohl für den ARM- als auch den Thumb-Betriebszustand gilt), ein privilegierter Prozessormodus existiert. Das Vorhandensein eines eigenen Stackpointers pro Betriebsmodus ermöglicht jedem einzelnen Betriebsmodus einen eigenen Stack. Im *Fast Interrupt Modus*[12] stehen außerdem die Register `R8` bis `R12` per *Register Shadowing* gesondert zur Verfügung, damit die schnelle Interruptverarbeitung beginnen kann, ohne

[10] `BL` oder `BLX`
[11] Current Program Status Register
[12] FIQ

Registerinhalte zu speichern oder wieder herzustellen.

Die Exceptions sind :

- Reset
- Attempted execution of an Undefined instruction
- Software interrupt (SWI)
- Prefetch Abort
- Data Abort
- IRQ
- FIQ

Wenn eine Exception auftritt, hält der ARM Prozessor die Ausführung des aktuellen Programms auf eine definierte Art und Weise an, schaltet auf den korresbondirerenden Prozessormodus um und beginnt die Ausführung an einem Exception-Vektor. Es gibt für jede Exception einen eigenen Exception-Vektor. Ein Betriebssystem kann im Initialisierungscode diese Exception-Vektoren setzen.

Interrupts

Der ARM7-Prozessor verfügt über zwei Arten von Interrupts. *Interrupt Request*(IRQ) und *Fast Interrupt Request*(FIQ). Die Verarbeitung von Interrupts wird einem speziellen Controller übertragen, dem *Vectored Interrupt Controller* (VIC). Der VIC nimmt die Interrupt Requests der verschiedenen Interruptquellen entgegen und teilt sie in drei verschiedene Kategorien. FIQ, *vectored* IRQ und *non-vectored* IRQ. Die höchste Priorität hat FIQ. *Nested* Interrupts sind auf dem ARM nur mit Workarounds realisierbar. Siehe hierzu [**?**] und insbesondere [30].

B Der verwendete Ethernet Controller

Der LPC2468 verfügt über einen 10/100 Mbps Ethernet-Controller, auch *MAC*[1] genannt, der zur Performancesteigerung *DMA*[2] verwendet. Der MAC und die CPU sind über einen dedizierten Bus (*AHB2*[3]) miteinander verbunden, über den auch ein dediziertes Ethernet-SRAM von 16 kB angebunden ist (siehe Abbildung B.2). Durch diese Anordnung ist es dem MAC möglich, als DMA-Master Daten vom Ethernet-SRAM zu laden, ohne dabei den Systembus (*ARM local bus*) zu blockieren. Aus Softwaresicht wird der Controller über 44 Register und die Datenfelder in speziellen Receive - und Transmitdesktiptoren im Speicher gesteuert. Abschnitt B beschäftigt sich eingehender mit diesen Datenstrukturen. Eine Übersicht über die Register des Controllers befindet sich in [33, S.212]. Der Controller ist vollständig kompatibel zum IEEE Standard 802.3, beherrscht Wake-On-Lan, verfügt über einen Empfangsfilter, automatische Kollisionserkennung und Retransmit, sowie optional automatische CRC-Checksummengenerierung und automatisches Frame Padding.

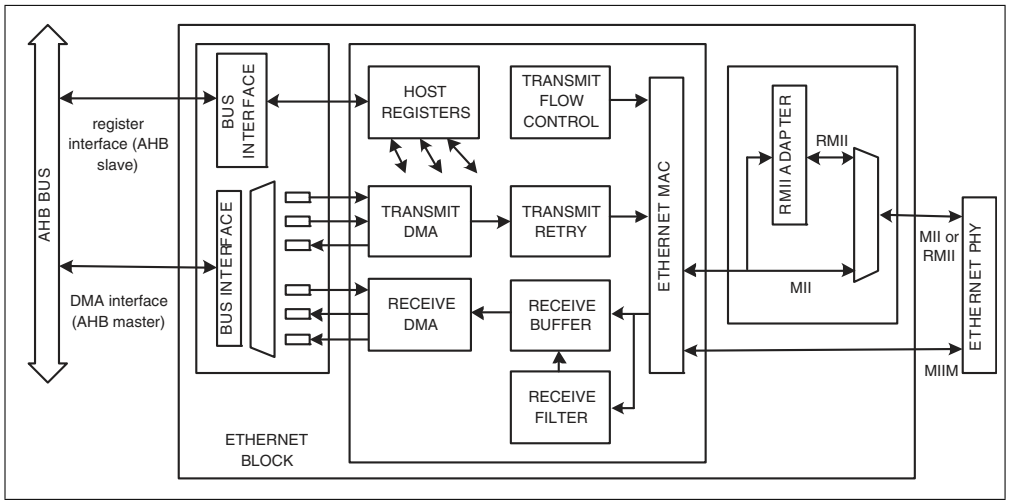


Abbildung B.1: Ethernet Block Diagramm ©NXP [33, S.207]

[1] Media Access Controller
[2] Direct Memory Access
[3] Advanced High-performance Bus 2

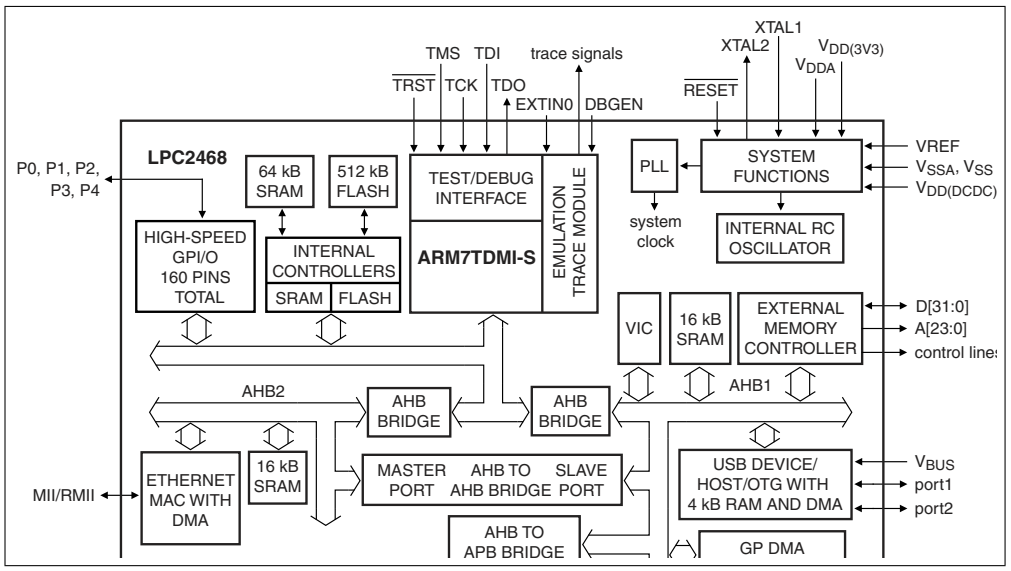


Abbildung B.2: LPC2468 Block Diagramm ©NXP [33, S.13]

Transmit- und Receivedeskriptoren

Das Senden und Empfangen von Netzwerkpaketen mit dem Ethernet-Controller geschieht über Arrays von sogenannten Transmit- und Receivedeskriptoren. Diese Arrays werden als Ring-Queues verwendet. Ein Deskriptor Besteht im Wesentlichen aus einem Pointer und einigen Kontrollinformationen. Neben jedem Deskriptor-Array gibt es ein StatusInfo-Array, welches ebenso viele Elemente wie das Deskriptor-Array haben muss. Diese Datenstrukturen müssen bei der Initialisierung der Treibersoftware des Ethernet-Controllers allokiert werden. Zu jedem Receivedeskriptor muß vor dem Aktivieren des Controllers außerdem je ein Datenpuffer allokiert werden, dessen Adresse und Größe im korrespondierenden Receivedeskriptor einzutragen ist. Damit der Controller mit diesen Arrays arbeiten kann, verfügt er über mehrere Register, welche die Basisadressen und die Größen der jeweiligen Arrays halten, sowie je ein *ProduceIndex* und ein *ConsumeIndex*-Register um den Füllstand der Queues zu speichern. Die Anzahl der Elemente der Deskriptorarrays ist zwischen 2 und 2^{16} frei wählbar und wird im jeweiligen `DeskriptorNumber`-Register abgespeichert. Jeder Deskriptor enthält neben einem Pointer auf den zugehörigen Datenpuffer auch Informationen über dessen Länge und Flags für die Interruptsteuerung in seinem CONTROL-Feld. In den Feldern des StatusInfo-Arrays legt der Controller weitere Informationen über die gesendeten bzw empfangenen Pakete ab.

Ein Deskriptorarray hat grundsätzlich immer einen Erzeuger und einen Verbraucher. Die Software ist Erzeuger von Transmitdeskriptoren und Verbraucher von Receivedeskriptoren, die Hardware ist Erzeuger von Receivedeskriptoren und Verbraucher von Transmitdeskriptoren. Der Erzeuger besorgt sich den über das jeweilige `ProduceIndex`-Register referenzierten nächsten frei-

en Deskriptor, füllt ihn mit Daten und gibt ihn an den Verbraucher, indem er das `ProduceIndex`-Register modulo der jeweiligen Arraylänge hochzählt. Der Verbraucher arbeitet mit den Informationen aus dem Deskriptor und dem referenzierten Datenpuffer und gibt ihn durch das Anpassen des jeweiligen `ConsumeIndex`-Registers zurück, damit er dem Erzeuger für weitere Aktionen wieder zur Verfügung steht. Der Besitzstatus der Deskriptoren in diesen Arrays wird durch die Positionen der Register `TXProduceIndex/TXConsumeIndex` bzw. `RXProduceIndex/RXConsumeIndex` geregelt, welche die jeweiligen Arrays logisch in zwei dynamisch fließende Hälften unterteilen. Die Queues sind leer wenn `ProduceIndex==ConsumeIndex` ist, sie sind voll, wenn `ProduceIndex==ConsumeIndex-1` ist. Nur der Besitzer eines Deskriptors liest oder schreibt dessen Inhalt. Abbildung B.3 zeigt ein Receivedeskriptor-Array und das dazu gehörende StatusInfo-Array.

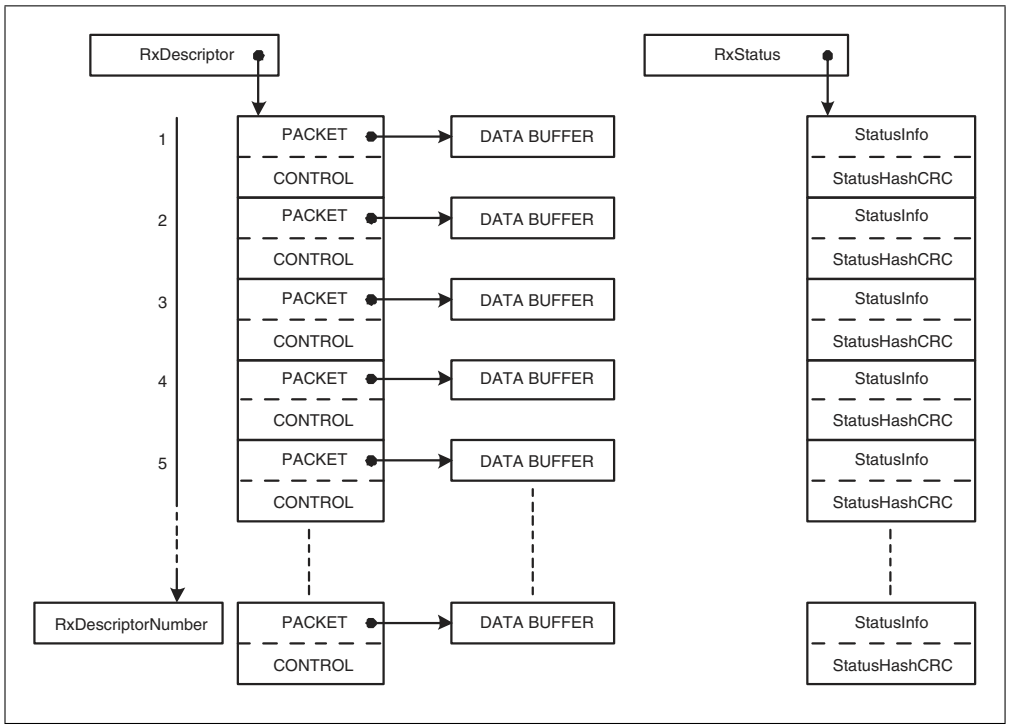


Abbildung B.3: Receive Descriptor memory layout ©NXP [33, S.236]

Direct Memory Access (DMA)

Der Ethernet-Controller beinhaltet zwei DMA-Manager zum Senden und Empfangen. Die DMA-Manager machen es möglich, Frames direkt vom Speicher zu lesen und in den Speicher zu schreiben, ohne dabei die CPU zu belasten oder dabei Interrupts zu erzeugen. Sie arbeiten mit den in Abschnitt B beschriebenen Deskriptor-Arrays. Ein Datentransfer ohne die Benutzung eines DMA-Managers ist nicht möglich. Sobald ein Paket durch die *PHY* empfangen oder durch die Software das Register `TXProduceIndex` verändert wird, wird automatisch der zugehörige DMA-Manager aktiviert.

Scatter/Gather DMA

Unter Scatter/Gather DMA versteht man DMA-Transfers, die in einem Arbeitszyklus Daten aus mehreren Adressbereichen sammeln bzw. Daten in mehrere Adressbereiche verteilen. Der Ethernet-Controller des LPC2468 verwendet diese Technik um das unnötige aktive Kopieren von Datenblöcken durch die CPU zu vermeiden. Beim Datenversand ist es zum Beispiel oft der Fall, dass gewisse Datenbereiche wie die Ethernetadresse sich nur selten ändern. Da solche Informationen im Programm oftmals in separaten Speicherbereichen verwaltet werden wäre es normalerweise notwendig, das Paket zunächst in einem Datenpuffer vorzubereiten, indem man die Ethernetadresse gefolgt von den Nutzdaten zusammenkopiert. Hier setzt Scatter/Gather DMA an. Anstatt die Bereiche aktiv durch die CPU in einen Speicher zu kopieren, überlässt man dies der DMA-Engine. Gesteuert wird dieser Vorgang über die Deskriptorarrays. Möchte man mehrere Datenbereiche einer Anwendung zu einem Paket zusammensetzen, benutzt man für jedes Segment einen einzelnen Deskriptor im Deskriptorarray. Die in den Transmitdeskriptoren referenzierten Daten werden nach Auslösen eines DMA-Transfers gesammelt (*Gather*) und versendet. Beim Datenempfang wird ebenfalls DMA verwendet. Empfängt der Ethernet-Controller ein großes Datenpaket, sucht er sich den nächsten freien Datenpuffer im Receivedeskriptor-Array. Reicht dieser nicht aus, um das gesamte empfangene Paket zu fassen, verwendet er auch den im nächsten Deskriptorfeld referenzierten Datenpuffer und verteilt das Paket auf die beiden Datenpuffer (*Scatter*). Die in den Deskriptoren enthaltenen Kontrollinformationen reichen aus, um die Daten weiterzuverarbeiten. Eine genauere Beschreibung des Datenempfangs befindet sich in Abschnitt B.4.

B.1 Die Ethernet-PHY

Der *MAC* verfügt über ein *MII* [4], über das ein *PHY* [5]-Chip angeschlossen werden kann. Über das MII lassen sich verschiedene Ethernet-Standards nutzen, hier können PHYs für 10 Base-T, 100 Base-TX, 100 Base-FX oder 100 Base-T4 angeschlossen werden. Das in dieser Arbeit verwendete HITEX®-Board LPC-Stick verwendet einen DP83848C PHYTER®von *National Semiconductors*, mit dem 10 Base-T und 100 Base-TX möglich sind.

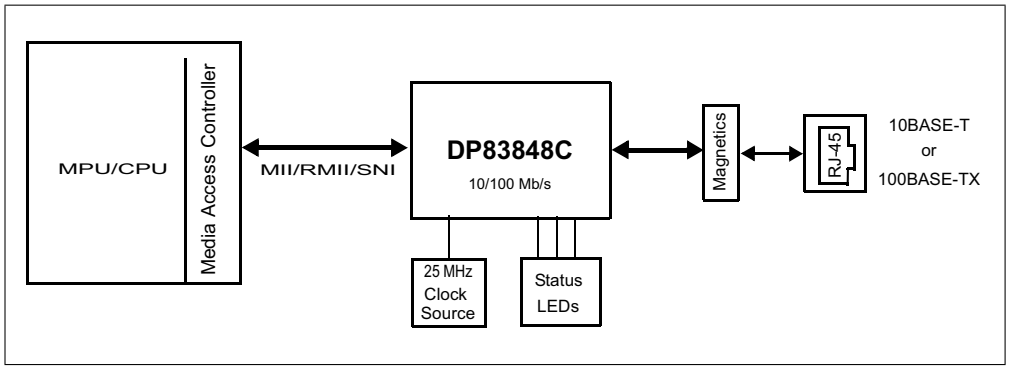


Abbildung B.4: Die Anbindung der PHY ©National Semiconductor [25, S.13]

B.2 Initialisierung des Ethernet-Controllers

Während der Initialisierung des Ethernet-Controllers muss die Software folgende Schritte erledigen:

- Die Host-Register (MAC1, MAC2, usw.) im Controller initialisieren
- Die Soft-Reset-Condition des Controllers entfernen
- MII oder RMII auswählen
- Den PHY über das MIIM-Interface konfigurieren
- Die Deskriptor-Arrays anlegen und die DMA-Engine initialisieren
- Die Transmit bzw Receive-Datenpfade aktivieren

Die Register `MAC1`, `MAC2` und `COMMAND` sind die wichtigsten Register der insgesamt 44 Register des Ethernet-Controllers. Hier können unter anderem die Datenpfade und der Empfangsfilter aktiviert, einzelne Komponenten des Controllers resettet, die automatische CRC- und PAD-generierung, der Betriebsmodus der Schnittstelle zum PHY (*MII* oder *RMII*) und der Duplexmodus eingestellt werden. Eine genauere Beschreibung aller Register des Ethernet-Controllers befindet sich in [33, S.212 ff].

[4] Media Independent Interface
[5] Ethernet Physical Sublayer

Über die Register MCFG, MCMD,MADR,MWTD,MRDD und MIND wird der an der MII-Schnittstelle des Ethernet-Controllers angeschlossene PHY konfiguriert. Die im PHY vorhandenen Register können über das MADR-Register adressiert werden, über das Register MWTD wird in das adressierte Register geschrieben, über MRDD kann gelesen werden. Über das MIND-Register kann der Status der MII-Schnittstelle abgefragt werden. Ein möglicher Schreibzugriff auf den PHY sieht so aus:

- Write 0 to MCMD
- Write PHY adress and register adress to MADR
- Write data to MWTD
- Wait for busy bit to be cleared in MIND

Mehr Informationen über die Konfigurationsregister des PHY in [25, S.38].

Der Empfangsfilter muß nicht unbedingt konfiguriert werden. In den Registern SA0-SA2 (*Station Address Register*) wird die Ehternet-Adresse (MAC-Adresse) des Controllers eingestellt.

Die Transmit- und Receive-DMA Engines werden initialisiert, indem die in Abschnitt B beschriebenen Register TXDescriptor, TxStatus, RxDescriptor und RxStatus mit den Basisadressen der zu allokierenden Deskriptor und StatusInfo-Arrays belegt werden. Die Deskriptor-Arrays müssen an 4-Byte-Adressgrenzen und die StatusInfo-Arrays an 8-Byte-Adressgrenzen angelegt werden, damit die DMA-Manager damit arbeiten können. Die Transmitdeskriptoren, die Receivedeskriptoren und die Receive-StatusInfo-Felder sind jeweils 8 Byte groß, die Größe eines Transmit-StatusInfo-Felds beträgt 4 Byte. Die Anzahl der Elemente der Deskriptorarrays ist zwischen 2 und 2^{16} frei wählbar, die minimale Größe dieser Arrays ist zwei, damit ein volles Array erkannt und von einem leeren Array unterschieden werden kann. Die Länge der Arrays wird in TXDescriptorNumber und RXDescriptorNumber in einer -1 Kodierung eingetragen (Hat das Array die Länge 8, wird in das Register 7 eingetragen). Zusätzlich zum Receivedescriptor-Array muß zu jedem einzelnen Receivedeskriptor ein Datenpuffer allokiert und dessen Adresse und Größe in die Datenfelder des Receivedeskriptors geschrieben werden. Die Größe der verwendeten Datenpuffer (auch Fragmentpuffer genannt) beträgt maximal 1 kB. Für das Transmitdeskriptor-Array werden keine Fragmentpuffer allokiert oder initialisiert, da sie im Gegensatz zu den Empfangs-Datenpuffern während des Betriebs dynamisch ausgetauscht werden. Die Basisadresse und die Länge der Deskriptor-Arrays darf während des Betriebs nicht geändert werden. Um die DMA-Engines letztendlich zu aktivieren, müssen *RXEnable* und *TXEnable* im COMMAND-Register gesetzt werden. Danach erst wird der Reveive-Datenpfad des Ethernet-Controllers durch das Setzen von *RECEIVE ENABLE* im MAC1-Register aktiviert. Diese Reihenfolge ist notwendig, damit eintreffende Datenpakete nicht während der Initialisierungsphase verarbeitet werden und unter Umständen einen Overflow in der Receive-DMA Engine erzeugen.

B.3 Datenversand

Folgende Schritte finden normalerweise statt, wenn ein Ethernet-Frame verschickt wird:

- Die Treibersoftware initialisiert einen Transmitdeskriptor
- Die Treibersoftware passt das `TXProduceIndex`-Register an
- TX-DMA liest das Transmitdeskriptor-Array
- TX-DMA überträgt die Daten
- TX-DMA schreibt das StatusInfo-Feld
- TX-DMA generiert ggf. einen Interrupt

Zunächst muß geprüft werden, ob ein Transmitdeskriptor frei ist. Wenn `TXConsumeIndex == TXProduceIndex-1` ist, ist das Deskriptor-Array voll. Ansonsten wird der Transmitdeskriptor an der Stelle `TxProduceIndex` verwendet. Der Packet-Pointer des Deskriptors wird auf die Adresse des Fragmentpuffers gesetzt. Im CONTROL-Feld des Deskriptors wird die Größe des Fragments eingetragen. Sollen für die Übertragung dieses Pakets abweichende Einstellungen für die CRC- oder PAD-Generierung gelten, kann dies ebenfalls hier eingestellt werden. Besteht das Paket aus nur einem Fragment, muß das '`Last`'-Bit gesetzt werden. Ein Paket kann aus mehreren Fragmenten zusammengesetzt werden, dies hängt direkt mit dem *Scatter/Gather-DMA* des Controllers zusammen. Das Zusammensetzen eines Pakets aus mehreren Fragmentpuffern geschieht über mehrere Transmitdeskriptoren. Möchte man ein Paket aus 3 Fragmentpuffern zusammensetzen, da beispielsweise die Ethernet-Adresse, ständig wiederholte Anwendungs-Header und die Nutzdaten in unterschiedlichen Puffern liegen, initialisiert man die Transmitdeskriptoren in der Reihenfolge, wie ihre Fragmentpuffer zusammengesetzt werden sollen und setzt nur bei dem letzten Transmitdeskriptor das '`Last`'-Bit im CONTROL-Feld.
Um einen Interrupt zu generieren, wenn ein Fragmentpuffer versendet wurde, kann für jeden Fragmentpuffer das '`Interrupt`'-Bit gesetzt werden.
Nach dem Schreiben des Transmitdeskriptors wird er durch das Inkrementieren von `TXProduceIndex` modulo `TXDescriptorNumber` an die Hardware übergeben. Um ein weiteres Paket zu versenden, muß nicht auf die Hardware gewartet werden, solange es noch freie Transmitdeskriptoren gibt.

Sobald der Transmit-Datenpfad eingeschaltet ist (was normalerweise einmalig in der Initialisierungsphase geschieht), vergleicht die DMA-Engine die Register `TXProduceIndex` und `TXConsumeIndex`. Wenn die beiden Register sich unterscheiden, beginnt die DMA-Engine damit, die Paketinformationen zu laden und das Paket zu versenden.

Zur Speicherung von Statusinformationen gibt es zu jedem Transmitdeskriptor ein StatusInfo-Feld im StatusInfo-Array. Im Statusinfo-Feld schreibt der Ethernet-Controller in Abhängigkeit vom Erfolg der Übertragung die Errorflags '`Error`', '`LateCollision`', '`ExcessiveCollision`', '`Underrun`', '`ExcessiveDefer`' und '`Defer`', sowie das '`CollisionCount`'-Feld. Die maximale Anzahl an Re-

transmits wird im *Collision Window / Retry Register* (`CLRT`) eingestellt. Wenn ein Fehler während des Sendens des ersten Fragments eines zusammengesetzten Pakets stattfindet, werden die verbleibenden Fragmente verworfen und das StatusInfo-Feld in die StatusInfo-Felder der verbleibenden Fragmente kopiert. Aus diesem Grunde muß nur das StatusInfo-Feld des letzten Fragments ausgewertet werden, da es auf diesem Wege den Erfolg der gesamten Übertragung darstellt. Der Status des jeweils letzten gesendeten Ethernet-Frames steht außerdem zu Debugzwecken in den Registern `TSV0` und `TSV1`, da die Kommunikation mit dem Ethernet-Controller ansonsten nur über die Deskriptoren stattfindet. Diese Register sollten nur gelesen werden, wenn die Transmit- und Receiveprozesse angehalten sind.

Nachdem das StatusInfo-Feld geschrieben wurde, gibt die Hardware den Transmitdeskriptor zurück indem sie das Register `TxConsumeIndex` modulo `TXDescriptorNumber` inkrementiert. Die TX-DMA Engine setzt das Senden von Paketen fort, bis das Transmitdeskriptor-Array leer ist und setzt danach das '`TxFinishedInt`'-Bit im Register `IntStatus`.

Das Versenden von Paketen kann jederzeit abgeschaltet werden, indem das '`TXEnable`'-Bit im `COMMAND`-Register gelöscht wird. Die DMA-Engine wird das Senden des aktuellen Pakets abschließen und danach anhalten.

Fehlerbehandlung

Wenn während der Übertragung ein Fehler auftrat, wird der TX-DMA Manager diesen über das StatusInfo-Feld berichten. Für die Fehlerfälle LateCollision, ExcessiveCollision und ExcessiveDefer werden zusätzlich das '`Error`'-Bit und das '`TXErrorInt`'-Bit im `IntStatus`-Register gesetzt. Im Falle eines '`Underrun`'-Fehlers wird das '`TXUnderrun`'-Bit im `IntStatus`-Register gesetzt. Wenn ein TXUnderrun-Fehler zusammen mit einem TXError berichtet wird, muß das Paket neu übertragen werden. Findet ein TXUnderrun-Fehler allein statt, war der AHB-Bus besetzt und die DMA Engine konnte das StatusInfo-Feld nicht rechtzeitig schreiben, bevor das Schreiben des nächsten StatusInfo-Felds an der Reihe war. In diesem Fall muß ein Soft-Reset im Ethernet-Controller über das '`Soft Reset`-Bit im Register `MAC1` durchgeführt werden.

B.4 Datenempfang

Ein einfaches Beispiel

Die Hardware empfängt Pakete vom Physikalischen Sublayer (*PHY*) und zunächst den im Ethernet-Controller vorhandenen Filter auf das empfangene Paket anwenden, falls dieser durch die Treiber-Software eingerichtet wurde. Falls das Paket nicht durch den Filter verworfen wird, liest die Hardware den nächsten freien Receivedeskriptor an der Stelle `RXProduceIndex` um die Adresse des nächsten freien Fragmentpuffers zu ermitteln. Die empfangenen Daten werden in den Fragmentpuffer geschrieben und zusätzliche Informationen über das empfangene Paket in das zugehörige StatusInfo-Feld des StatusInfo-Arrays geschrieben. Das empfangene Paket wird an die Software übergeben, indem das Register `RXProduceIndex` auf das nächste Feld im RXDeskriptor-Array gesetzt und optional ein `RxDone`-Interrupt generiert wird.

Ein komplexeres Beispiel

- Die Treibersoftware aktiviert den Receive-Datenpfad und den RX-DMA
- Ein Paket trifft ein
- RX-DMA liest das Receivedeskriptor-Array
- RX-DMA überträgt das Paket in den Fragmentpuffer des nächsten freien Receivedeskriptors
- RX-DMA schreibt das StatusInfo-Feld
- RX-DMA passt das Register `RXProduceIndex` an
- RX-DMA generiert ggf. einen Interrupt

Ist das '`RXEnable`'-Bit im `COMMAND`-Register gesetzt, beginnt die RX-DMA Engine Receivedeskriptoren zu einzulesen. Dies geschieht schon bevor Pakete eintreffen (Deskriptor Prefetching). Das Einlesen geschieht blockweise, der Controller lädt alle freien Deskriptoren zwischen `RxConsumeIndex` und `RXProduceIndex-1`. Das blockweise Einlesen von Deskriptoren minimiert die Speicherzugriffe. Die gelesenen Daten werden gepuffert und nach Bedarf verwendet.

Nach dem Lesen der Deskriptoren wartet die RX-DMA Engine auf Pakete, die den Empfangsfilter passieren. Frames, die nicht den Filterkriterien entsprechen werden verworfen, ohne in den Speicher übertragen zu werden. Passiert ein Paket den Empfangsfilter, wird es in den assoziierten Fragmentpuffer des nächsten freien Deskriptors übertragen. Wenn ein Paket empfangen wird welches größer ist als der Fragmentpuffer, wird das Paket im nächsten Deskriptor fortgesetzt und im letzten Deskriptor das '`Last`'-Bit im StatusInfo-Feld auf '1' gesetzt. Die eigentlich genutzte Kapazität eines Fragmentpuffers wird im Feld '`RxSize`' im StatusInfo-Feld gespeichert. Eine gängige Strategie zum Speicher-

layout ist, die Fragmentpuffer im Speicher direkt nacheinander anzulegen. Ein über mehrere Deskriptoren empfangenes Paket kann so durch die Software in einem sequenziellen Lesezugriff aus dem Speicher gelesen werden.

Nachdem der RX-DMA ein Fragment übertragen hat, schreibt er die Statusinformationen und die CRC in das StatusInfo- und das StatusHashCRC-Feld. Das `RXProduceIndex`-Register wird erst aktualisiert, wenn die Daten vollständig in den Fragmentpuffer geschrieben wurden. Ist das StatusInfo-Feld geschrieben, übergibt der Ethernet-Controller das empfangene Fragment an die Software, indem er das `RXProduceIndex`-Register anpasst und das '`RXFinished`'-Bit im Register `IntStatus` setzt. Der Fortschritt des DMA Managers kann auch beobachtet werden, indem das `RXProduceIndex`-Register ausgewertet wird.

Ist das Fragment das Letzte einer Datenübertragung oder passt das gesamte Frame in einen Fragmentpuffer, werden die Error-Flags im StatusInfo-Feld gesetzt. Nur der letzte Deskriptor enthält diese Informationen. Eine Übersicht über das StatusInfo-Feld für den Empfang befindet sich in [33, S. 238, Tab. 226]. Der Status des jeweils letzten gesendeten Ethernet-Frames steht außerdem zu Debugzwecken im Register `RSV`, da die Kommunikation mit dem Ethernet-Controller ansonsten nur über die Deskriptoren stattfindet. Dieses Register sollten nur gelesen werden, wenn der Empfangsprozess angehalten ist.

Fehlerbehandlung

Ist das Receivedeskriptor-Array voll, setzt der Controller das '`RxFinished`'-Bit im Register `IntStatus`. Trifft in dieser Situation ein weiteres Paket ein, wird es verworfen, Overflow error gesetzt und ein Interrupt erzeugt.

Der Empfangsprozess kann verschiedene Fehler erzeugen. AlignmentError, RangeError, LengthError, SymbolError, CRCError, Overrun und NoDescriptor. Alle haben korrespondierende Bits im StatusInfo-Feld. Zusätzlich dazu gibt es ein Error-Feld, welches gesetzt wird wenn AlignmentError, RangeError, LengthError, SymbolError oder CRCError gesetzt sind. Fehler werden außerdem im `IntStatus`-Register propagiert. Das '`RXError`'-Bit wird gesetzt, wenn AlignmentError, RangeError, LengthError, SymbolError, CRCError oder NoDescriptor gesetzt sind. Dies sind sogenannte 'nichtfatale' Fehler.

B.5 Treiber

In dieser Arbeit ist ein Treiber von Hitex zum Einsatz gekommen. Die Initialisierung des Netzwerkchips ist in Abbildung B.5 grob dargestellt. Der Treiber (`ethernet_lpc24xx.c`) befindet sich auf der CD.

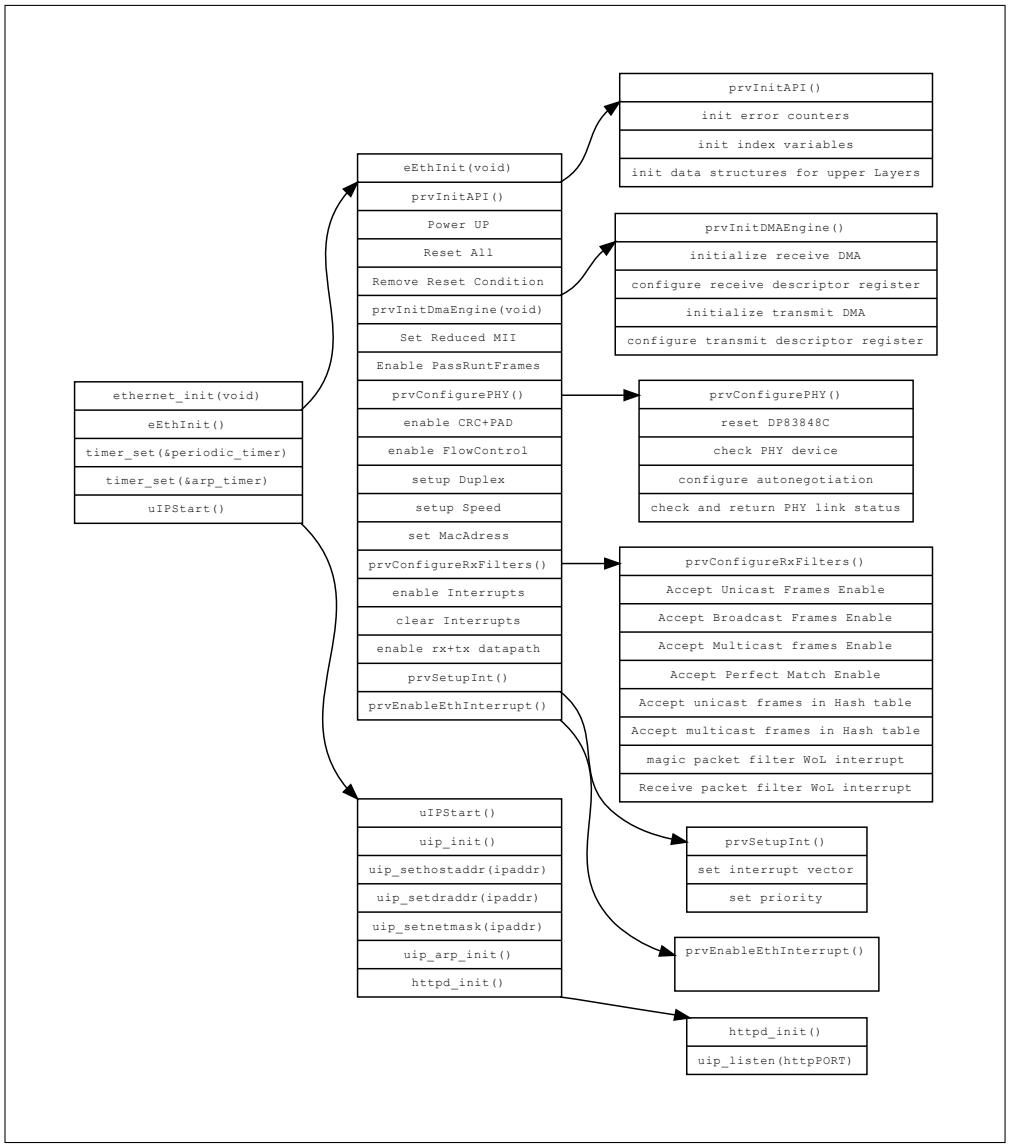


Abbildung B.5: Die Initialisierung des Netzwerkchips

C Messung bei 48MHz

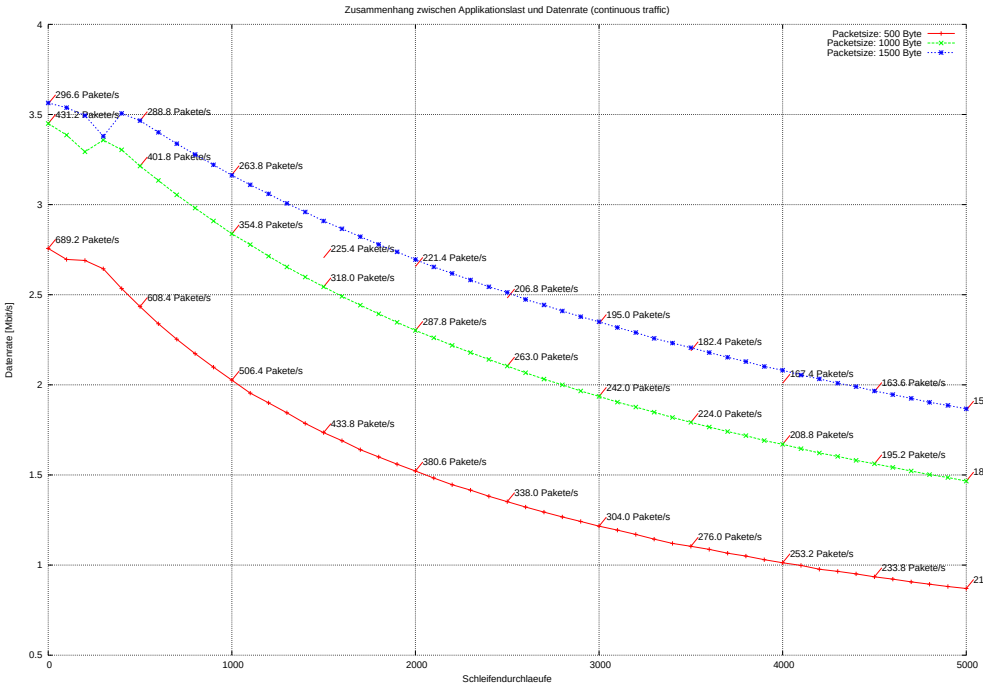


Abbildung C.1: Applikationslast bei 48MHz

Diese Statistik wurde mit der gleichen Technik wie auf Seite 89 erstellt. Die CPU wurde mit 48MHz betrieben. Der Test zeigt die Datenraten bei den Paketgrößen 500 Bytes, 1000 Bytes und 1500 Bytes.

Literaturverzeichnis

[1] *Philips NXP Semiconductors*. http://www.nxp.com.

[2] *Programm DIAdem*. http://www.ni.com/diadem/d.

[3] *Qt Cross Platform*. http://trolltech.com/products/qt.

[4] *Wikipedia: Agilent VEE*. http://de.wikipedia.org/wiki/Agilent_VEE.

[5] *Wikipedia: Feldbus*. http://de.wikipedia.org/wiki/Feldbus.

[6] *Wikipedia: LabVIEW*. http://de.wikipedia.org/wiki/LabVIEW.

[7] *Wikipedia: Messtechnik*. http://de.wikipedia.org/wiki/Messtechnik.

[8] *Wikipedia: OSI-Modell*. http://de.wikipedia.org/wiki/OSI-Modell.

[9] *Wikipedia: Regelkreis*. http://de.wikipedia.org/wiki/Regelkreis.

[10] *Wikipedia: Regelungstechnik*. http://de.wikipedia.org/wiki/Regelungstechnik.

[11] *Wikipedia: Regler*. http://de.wikipedia.org/wiki/Regler.

[12] *Wikipedia: TCP/IP-Referenzmodell*. http://de.wikipedia.org/wiki/TCP/IP-Referenzmodell#TCP.2FIP-Referenzmodell.

[13] Defense Advanced Research Projects Agency. *RFC 793: Transmission Control Protocol - DARPA INTERNET PROGRAM – PROTOCOL SPECIFICATION*, September 1981. http://tools.ietf.org/html/rfc793.

[14] European Computer Manufacturers Association. *Standard ECMA-262 - ECMAScript Language Specification*, third edition, December 1999. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

[15] Adam Dunkels. *Design and Implementation of the lwIP TCP/IP Stack*, February 2001. `http://www.sics.se/~adam/lwip/doc/lwip.pdf`.

[16] Adam Dunkels. *Full TCP/IP for 8-Bit Architectures*, 2003. `http://www.sics.se/~adam/mobisys2003.pdf`.

[17] Adam Dunkels. *The uIP Embedded TCP/IP Stack (Reference Manual)*, June 2006. `http://www.sics.se/~adam/download/uip-1.0-refman.pdf`.

[18] Sun Microsystems inc. *The Java Tutorials: 2D Graphics*. `http://java.sun.com/docs/books/tutorial/2d/TOC.html`.

[19] Sun Microsystems inc. *The Java Tutorials: Using Layout Managers*. `http://java.sun.com/docs/books/tutorial/uiswing/layout/using.html`.

[20] Sun Microsystems inc. *Trail: The Reflection API*. `http://java.sun.com/docs/books/tutorial/reflect/index.html`.

[21] Sun Microsystems inc. *Using Swing Components*. `http://java.sun.com/docs/books/tutorial/uiswing/components/componentlist.html`.

[22] ARM Limited. *ARM7TDMI-S Technical Reference Manual (Rev. 3)*, September 2000. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0084f/DDI0084.pdf`.

[23] ARM Limited. *ARM PrimeCell™ Vectored Interrupt Controller (PL192) Technical Reference Manual*, December 2002. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0273a/DDI0273.pdf`.

[24] ARM Limited. *Arm Architechture Reference Manual*, July 2005. `http://www.arm.com/miscPDFs/14128.pdf`.

[25] ARM Limited. *DP83848C PHYTER®- Commercial Temperature Single Port 10/100 Mb/s Ethernet Physical Layer Transceiver*, May 2008. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0273a/DDI0273.pdf`.

[26] Advanced Risc Machines Ltd. *Application Note 93 - Benchmarking with ARMulator*, 2002. `http://infocenter.arm.com/help/topic/com.arm.doc.dai0093a/DAI0093A_benchmarking_appsnote.pdf`.

[27] Advanced Risc Machines Ltd. *ARM Corporate Backgrounder*, August 2005. `http://www.arm.com/miscPDFs/3822.pdf`.

[28] Michael J. Pont. *Patterns for Time-Triggered Embedded Systems*, 2001. http://www.tte-systems.com/downloads/pttes_0408a.pdf.

[29] J. Postel. *RFC 768: User Datagram Protocol*, August 1980. http://tools.ietf.org/html/rfc793.

[30] NXP Semiconductors. *AN10381 - Nesting of interrupts in the LPC2000*, June 2005. http://www.standardics.nxp.com/support/documents/microcontrollers/pdf/an10381.pdf.

[31] NXP Semiconductors. *AN10381 - Nesting of interrupts in the LPC2000*, June 2005. http://www.standardics.nxp.com/support/documents/microcontrollers/pdf/errata.lpc2468.pdf.

[32] NXP Semiconductors. *AN10404 - Initialization code/hints for the LPC2000 family*, November 2005. http://www.standardics.nxp.com/support/documents/microcontrollers/pdf/an10404.pdf.

[33] NXP Semiconductors. *LPC24xx User Manual Rev. 01.05*, December 2007. http://www.standardics.nxp.com/support/documents/microcontrollers/pdf/user.manual.lpc24xx.pdf.

[34] NXP Semiconductors. *LPC2468 Erratasheet V1.6*, June 2008. http://www.cs.ucr.edu/~amitra/sdcard/ProdManualSDCardv1.9.pdf.

[35] W. Simpson. *RFC1994: PPP Challenge Handshake Authentication Protocol (CHAP)*, aug 1006. http://tools.ietf.org/html/rfc1994.

[36] Treck. *Treck TCP/IP User Manual*, 2004. http://www.treck.com/PDF/truser40e.pdf.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 15. August 2008
Ort, Datum

Unterschrift