Oliver Jänicke

Design und Entwicklung einer Ajax – basierten Prozessvisualisierung für Echtzeitanwendungen

Bachelorarbeit eingereicht im Rahmen des Studiengangs Informatik
Studiengang Technische Informatik
am Fachbereich Elektrotechnik und Informatik der Hochschule für Angewandte Wissenschaften der Freien und Hansestadt Hamburg

Betreuender Prüfer : Prof. H. H. Heitmann
Zweitgutachter : Prof. B. Buth

Abgegeben am : 14. 12. 2007

Zusammenfassung

In dieser Arbeit wird eine Ajax-basierte Kommunikationsstrecke zur Prozessvisualisierung von Echtzeit-Prozessen realisiert. Dazu wird ein Server entworfen und realisiert, der Anwenderprogramme integriert, die mit Echtzeitprozessen kommunizieren können. Die erfragten Prozessdaten können Clientseitig mit einem Standard – Webbrowser dargestellt werden. Es können so dynamische Tabellen und Grafiken über Prozessverläufe erstellt werden, aber auch mit Parametern auf Echtzeit-Prozesse Einfluss genommen werden.

Abstract

In this bachelor thesis a ajax based communication path for process visualisation of realtime processes is developed. A server is designed and created which integrates user programs, which are capable of communicating with realtime processes. The requested process data can be displayed on a standard webbrowser. Dynamic tables and graphics of process progress can be generated, but it is possible to control realtime processes with parameters.

Danksagung

An dieser Stelle möchte ich vorab einige Personen lobend erwähnen ohne die diese Arbeit in dieser Form nicht entstanden wäre. Da wäre als erstes mein Betreuer Prof. H. H. Heitmann zu nennen, durch den erst mein Interesse für dieses Thema geweckt wurde. Des weiteren gilt mein Dank meinem Zweitgutachter Prof. Betina Buth, deren Ratschläge bei der Analyse und den Performance – Tests der Beispielanwendung, unverzichtbar für die realistische Einschätzung dieser Arbeit gewesen ist. Weiterhin einigen Mitstudenten, die immer einige Minuten und mehr für mich übrig hatten und es somit mehr als verdient haben, hier erwähnt zu werden. Z.B. Alexander Kant, der mir mit seiner mehrjährigen Erfahrung in HTML – und Scriptsprachen viel Rechercheарbeit und Übungszeit erspart hat. Oder auch Andreas Liedke, mit dem ich versucht habe einen kleinen Linux – Echtzeitkernel auf den verwendeten Beispielrechner zu portieren. Nach mehreren Fehlschlägen und stundenlanger manueller Konfigurationsarbeit ist es gelungen, Linux RTAI als Demonstrationsplattform einzurichten. Ohne sein Insider – Wissen hätte dieser Punkt gestrichen werden müssen und die Simulation wäre nicht so realistisch ausgefallen. Nochmals vielen Dank Andreas. Auch Bezad Daghigh muss hier erwähnt werden, der im Rahmen seines Praktikums auf einem ganz ähnlichen Gebiet seine Abschlussarbeit schreibt und mit dem ich immer Erfahrungen austauschen konnte. Und dann danke ich natürlich meiner Familie, die mich die ganzen Jahre unterstützt hat und immer dann zur Stelle war, wenn das Geld mal wieder sehr knapp wurde. Zum Schluss wäre da noch die Stadt Hamburg zu nennen, die mir mit ihrem Hochschulgesetz ein Studium an der HAW ermöglicht hat, obwohl mir das nötige Abitur fehlt.

Nicht danken möchte ich der Fachbibliothek der HAW die eigentlich den Namen Fachbibliothek nicht verdient. Um dynamische Webanwendungen zu erstellen gibt es mehrere Möglichkeiten. Eine davon nennt sich Ajax (dazu später mehr). Unter dem Stichwort Ajax lieferte die Datenbank der Bibliothek 21 Treffer. Von diesen 21 Treffern war kein einziges Buch ausleihbar. Eine weitere Möglichkeit wäre SVG (Scalable Vector Graphics) gewesen. Hier lieferte die Datenbank 4 Treffer. Überflüssig zu erwähnen, dass auch hier keines der Bücher vorrätig war. In Anbetracht der Tatsache, dass hohe Studiengebühren verlangt werden, hätte ich wenigsten erwartet, dass der Bibliothek dieser Umstand unangenehm, ja vielleicht sogar peinlich wäre. Weit gefehlt. Im Gegenteil; man begreift Mangel wieder als Chance für ein tolles Marketingkonzept. Jeder, der jetzt ein nicht vorrätiges Buch benötigt, muss Gebühren bezahlen. Das Konzept ist ebenso dreist wie genial, gehört aber eigentlich strafrechtlich verfolgt. Das Konzept ist aber noch ausbaufähig. Wenn sämtliche Angestellten alle Bücher ausleihen, wäre jedes Buch gebührenpflichtig. Der Weg hin zu einem zukunftsorientierten und “modernen“ Hochschulwesen heißt eines mit Sicherheit nicht; dass es für die Studenten besser wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

In dieser Arbeit wird eine webbasierte Client / Server – Anwendung entwickelt, die das Verhalten externer Prozesse grafisch darstellen kann. Als Clientseite dient ein Webbrowser, der über eine Arbeitsseite Daten vom Server anzeigt, sowie über Eingabefelder den Server mit Parametern versorgen kann.

Der Server ist Teil eines Industrierechners und steht über interne Anwendungsprogramme mit den Prozessen der zu steuernden Anlage in Verbindung. Diese Prozesse können Sensordaten, wie Temperatur, Geschwindigkeit, Druck usw., in Registern hinterlegen, auf die der Server lesend zugreifen kann. Parameter, die der Server vom Client übergeben bekommt, können im Gegenzug das Verhalten der Anlage beeinflussen.

In einer derartigen Umgebung haben die Komponenten ganz bestimmte Voraussetzungen zu erfüllen. Der Arbeitsspeicher derartiger Anlagen ist in der Regel begrenzt. Ein Großteil der industriellen Anwendungen müssen zeitkritischen Anforderungen bestehen, die somit im Wiederspruch zum Internet stehen. Der Server wird also nur als normaler Prozess mit einer geringen Priorität laufen können. Es ist daher nötig, einen schnellen und ressourcenschonenden Server zu entwickeln, der speziell auf die Anforderungen vor Ort zugeschnitten ist. Die Struktur muss aber generalisiert werden. Es dürfen bei der Portierung auf andere Plattformen nur die Systemfunktionen ausgetauscht werden. Auf die interne Funktionalität darf dies keinen Einfluss haben.

Die Clientseite ist zwar weniger problematisch, sollte aber trotzdem Zustandsänderungen so schnell wie möglich auf dem Bildschirm darstellen können. Die Architektur eines Webbrowsers muss erst einmal als gegeben hingenommen werden. Es wird also darauf ankommen, möglichst viel Funktionalität dem Browser zu überlassen, damit sich der Server nur mit der Beschaffung der Daten beschäftigen muss. Um dabei so schnell wie möglich zu sein, werden die Anwendungen in den Server integriert und auf das Starten eines Anwendungsprozesses verzichtet.

Am Ende der Arbeit steht ein Server, der auf Anfrage Daten von Echtzeitprozessen bekommt und auf der Clientseite ein Standardbrowser, der diese Daten mit modernen Techniken darstellen kann, während bei vorhandene Lösungen, spezielle Diagnosesoftware installiert werden muss. Es wird so ein Maximum an Flexibilität erreicht, bei gleichzeitiger Kostenminimierung.

2 Stand der Technik

In einem Fachartikel der Konradin – Mediengruppe [1] wird unter dem Begriff "Echtzeit - Surfing" eine Technologie vorgestellt, mit der Scada-Daten über Intranet und Internet übertragen werden können. Durch eine Client/Server-Architektur auf Basis von Windows NT in Kombination mit einer "leistungsfähigen HMI-Software" sollen solche Systeme eine nahtlose Verbindung zwischen Produktion und Bediener anbieten. Als Ergebnis liefern Systeme wie Cimplicity HMI von GE Fanuc bereits optimierte Lösungen für unterschiedliche Anwendungsbereiche, die von kleinen, unabhängigen Prozessen bis hin zu anlagenweiten Systemen für Kraftwerke, Chemiefabriken oder weltweit operierende Automobilhersteller reichen. Erforderlich soll hierfür lediglich ein PC und ein Standardbrowser sein.

Der Fachartikel beschreibt diese Innovation folgendermaßen:

Zum ersten Mal können beliebige Produktionsdaten, darunter dynamische Grafiken, Punktdaten und ActiveX-Objekte, abgerufen und sogar interaktiv bearbeitet werden, auch vom anderen Ende der Welt aus. Durch die direkte Echtzeit-Verfügbarkeit von Prozessdaten, Qualitätsdaten, Lagerbeständen, Energieverbrauch, Produktionsauslastung, Produktionsterminen oder beliebiger anderer Daten des Scada-Systems können Entscheidungsprozesse verbessert, Produktionsabläufe und das Lagerwesen optimiert sowie generelle Managementaufgaben erleichtert werden. Statt auf Berichte zu warten, lassen sich Daten von unterschiedlichen Prozessen, Fertigungsstraßen oder sogar verschiedenen Werken aus aller Welt in Echtzeit unmittelbar miteinander vergleichen und Entscheidungsprozesse somit verbessern und beschleunigen. So basiert die WebView-Software von GE Fanuc [2] auf Thin-Client-Technologie und bietet eine Lösung an, die praktisch keine Anschaffungskosten verursacht. WebView wird auf dem Webserver installiert und wandelt automatisch Scada-Masken in Standard-HTML-Seiten um, die direkt vom Anwender-PC mit einem Standardbrowser wie Internet Explorer von Microsoft oder Netscape Navigator abgerufen werden können. Da die gesamte Aufbereitung der Scada-Masken auf dem Webserver erfolgt, muss auf dem PC keinerlei Scada-Software installiert, konfiguriert oder gepflegt werden. Die Realisierung ist somit ausgesprochen einfach und kostengünstig. Die Lizenzverwaltung für ein Produkt wie WebView ermöglicht den Zugriff durch mehrere Benutzer - sobald sich einer aus dem System abgemeldet hat, wird die Lizenz für den nächsten Benutzer ohne Zusatzkosten wieder freigegeben. Die Gesamtzahl der berechtigten Benutzer ist nicht beschränkt. Der Webserver sendet die angeforderten Masken oder Daten automatisch als reine HTML-Seiten an den Browser; Programmierung, Bildschirmaufbau und auch die Bearbeitung von HTML-Dateien entfallen. Der gesamte Vorgang ist für den Anwender absolut integriert und transparent. Mit diesen neuen Webtools haben viele Benutzer Zugang zu anspruchsvollen dynamischen Grafiken, Produktionsstatistiken und anderen aufschlussreichen Informationen wie Alarmmeldungen und Prozesswerte.

Soweit die Fachpresse. Welcher Typ von Webserver diese Aufgabe bewältigt und welche Kosten auf den Nutzer zukommen, bleibt offen. Auch mit welcher Technik die vorgefertigten HTML-Dateien ihre Dynamik bekommen sollen, ohne sie zu bearbeiten bzw. zu aktualisieren, bleibt ihr Geheimnis. Des Weiteren wird der Begriff "Echtzeitverfügbarkeit" in Verbindung mit Qualitätsdaten, Lagerbestände und Energieverbrauch verwendet was nicht so recht schlüssig scheint. Lagerbestände werden wohl eher mit einer Datenbank oder Terminplaner verwaltet und Statistiken wie Energieverbrauch und Qualitätsdaten sind wohl eher keine zeitkritischen Daten. Offenbar werden hier Fachbegriffe zu Marketingzwecken missbraucht was nicht sehr vertrauenswürdig erscheint. Mehr Informationen sind auf der Homepage von GE Fanuc und Konradin zu finden.

Siemens bietet einen WinCC/Web Navigator an [3], der die Möglichkeit bietet, Anlagen über das Internet oder firmeninterne Intranet bzw. LAN zu bedienen und zu beobachten, ohne dass Änderungen am WinCC-Projekt notwendig sind. Damit ergeben sich dieselben Darstellungs-, Bedien- und Zugriffsmöglichkeiten auf Archive wie bei Bedienstationen vor Ort.

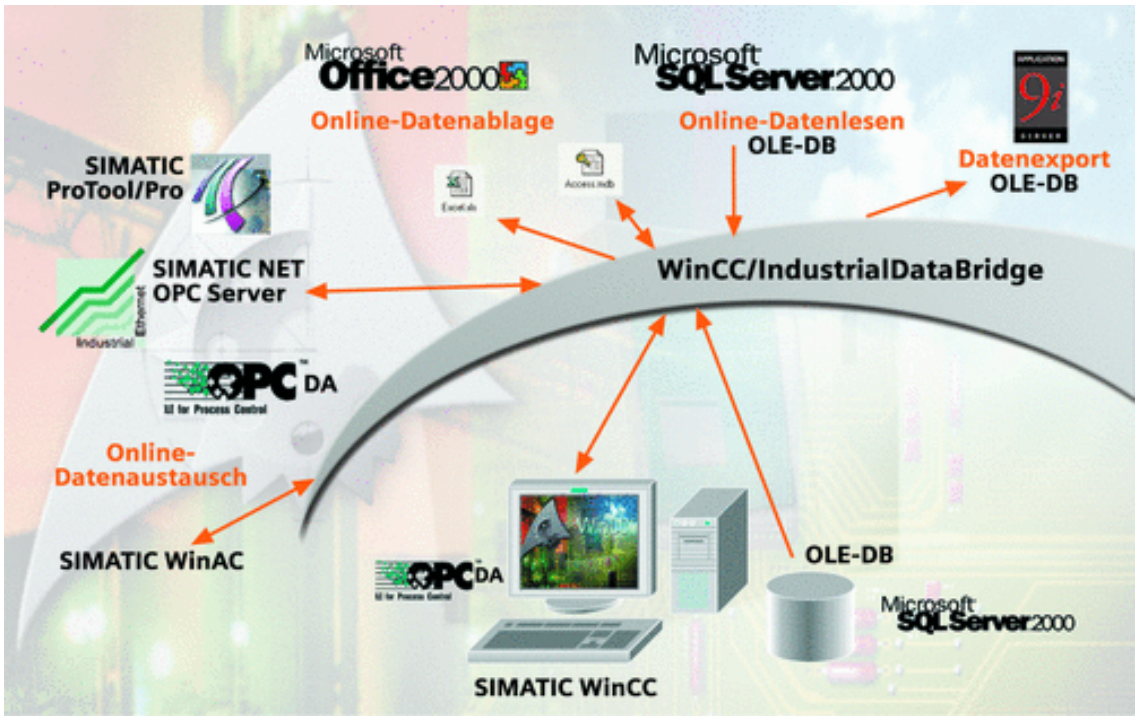


Abbildung 1: Kommunikationsschema von Siemens

Dazu muss allerdings auf einem WinCC Einplatzsystem oder Server der Web - Navigator Server installiert werden. Auf einem beliebigen Windows-Rechner muss ein Web Navigator Client eingerichtet sein. Dieser ermöglicht das Bedienen und Beobachten eines laufenden WinCC-Projektes über den MS Internet Explorer ab Version 6.0, ohne dass dazu das WinCC-Basissystem auf dem Rechner vorhanden sein muss.

Ein solcher Web-Server kann auch auf jedem WinCC (SCADA)-Client eingerichtet werden. Damit hat ein am Web-Server angeschlossener Web-Client von überall auf der Welt Zugriff auf die Projekte aller bis zu 12 (redundanten) WinCC-Server in einer Anlage. Hierbei schaltet der Web - Client auch transparent zwischen redundanten, unterlagerten WinCC Servern um. Wird der Browser am Web - Client mehrfach gestartet, ist sogar der Blick in mehrere Anlagen also auch mehrere Web-Server gleichzeitig möglich.

Bei dieser Lösung fällt auf, dass spezielle Server und Clients installiert sein müssen, um über das Web auf industrielle Anlagen zugreifen zu können. Das nur der Internet Explorer als Client in Frage kommt, ist sicherlich auch nicht gerade ein Pluspunkt. Es stellt also eine hersteller – bezogene Lösung dar und solche Lösungen bieten viele Hersteller von Steuerungstechnik an.

Ein letztes Projekt, was hier erwähnt werden soll, ist die WebNet Internet Leitstelle [4].

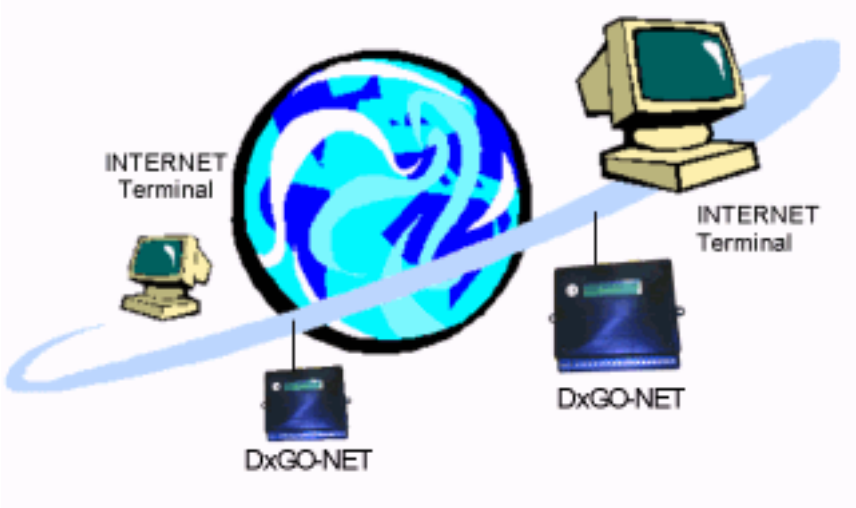


Abbildung 2: Kommunikationsschema WebNet

Sie erlaubt Prozesse über Standard Internet-Browser zu visualisieren und zu steuern. Die Leitstellenfunktionalität ist auf diese Weise an jedem beliebigen Ort weltweit verfügbar. Passwortschutz und Benutzerprofile erlauben nur ausgewählten Personen den Zugriff auf für sie bestimmte Informationen. Die Darstellung der Prozesse erfolgt möglichst originalgetreu in interaktiven Prozessbildern. Buttons dienen zur Aktivierung von Pumpen und Stellgliedern. Das Antwortzeitverhalten wird durch das System und die Art der Inter-/Intranetanbindung bestimmt. Typische Antwortzeiten liegen im Sekundenbereich oder darunter und sind damit völlig ausreichend für Anwendungen z.B. im Wasser- oder Wärmebereich. Neben den Prozessen können Protokollauswertungen durchgeführt und angezeigt werden. Loggerfunktionen werden ebenfalls zur Verfügung gestellt. Alle Informationen werden ausschließlich auf dem Server gehalten. Die Ausgabe ausgewählter Daten in Status-/Tages- oder Monatsprotokollen ist möglich.

Zur Erstellung eigener Bilder oder Masken steht ein komfortabler Editor mit online - Testmöglichkeiten bereit. Wer also nicht aus dem Wasser - oder Wärmebereich kommt, muss sich seine Seiten selbst programmieren. Auf welchen Plattformen der Server überall läuft wird verschwiegen. Stattdessen bietet man eine Systemlösung an, die Soft – und Hardware beinhaltet.

Es scheint fast so, als gäbe es genau so viele Lösungen, wie Hersteller. Und genau das ist in der heutigen Zeit nicht gewünscht. In Zeiten der Globalisierung, wo Betriebsteile und ganze Unternehmen den Besitzer wechseln, nimmt die Vielfalt der IT – Infrastrukturen innerhalb eines Unternehmens, durch diese Art herstellerbezogener Lösungen, immer mehr zu. Jeder bringt sein eigenes System in das neue Unternehmen ein. In diesem Fall ist Vielfalt unerwünscht. Sie führt zu großer Unübersichtlichkeit. Spezialisten für jedes System werden benötigt. Die Integration von anderen Maschinen und Anlagen ist kaum noch möglich und wenn, dann nur mit aufwendigen und teuren Soft – und Hardwarekomponenten mit dem Ergebnis, dass alles immer komplexer aber nichts besser wird. Im Gegenteil; die Kosten steigen und die Ausfallwahrscheinlichkeit nimmt zu. Und eine Schnittstelle, mit der der Server Daten aus echtzeitfähigen Programmteilen erfragen kann, erwähnt keiner.

Dabei sind alle Voraussetzungen wie Internet, netzwerktaugliche Büro – und Anlagenhardware sowie Kommunikationsprotokolle seit langem vorhanden. Was fehlt, ist ein spezieller Server, der auf systemkritische Daten Zugriff hat und diese Daten in Formate konvertiert, die von Clients dargestellt werden können, die auf jedem Rechner standardmäßig installiert sind (Webbrowser).

Es stellt sich also die Frage: Gibt es vielleicht schon Server die diese Aufgaben übernehmen oder wenigstens leicht an diese angepasst werden können ? Und welche Eigenschaften müsste er besitzen, damit er sich überhaupt für dieses Umfeld als tauglich erweist ?

3 Verschiedene Server im Vergleich:

Folgende Eigenschaften und Kriterien werden von mir als minimale Bedingung vorausgesetzt:

1. Er sollte so ressourcenschonend wie möglich sein
2. Er sollte möglichst plattformunabhängig sein
3. Er sollte möglichst schnell sein, auch wenn dies zu Lasten des Komforts geht
4. Er soll modular aufgebaut sein und dadurch beliebig erweiterbar.
5. Und er muss sich auf Echtzeit – Betriebssysteme portieren lassen.

Der Server, der benötigt wird, muss eigentlich nur Dateien liefern können, Daten vom Client entgegennehmen und Antwort – Daten zurück zum Client schicken können. Die Daten bestehen nur aus primitiven Datentypen wie Text und Zahlen, eben Parameter, die den Zustand einer Anlage charakterisieren bzw. darstellen. Er soll keine rechenintensiven Grafiken erstellen können.

Die Darstellung der gelieferten Daten, wird dem Lieblingsbrowser des Anwenders überlassen. Als zweites muss geklärt werden, ob bereits Server existieren, die diese Kriterien erfüllen und wenn nicht, ob wenigstens Teile von “Open Source“ - Projekten verwendet werden können. Das würde auf jeden Fall viel Programmierarbeit sparen. Es sollen nun einige Webserver aufgezählt werden und die vermeintlich Wichtigsten etwas näher auf die geforderten Eigenschaften hin untersucht werden. Wikipedia liefert eine Liste [5], von der die vermeintlich Geeignetsten auf ihre Tauglichkeit geprüft werden sollen. Alle sind auf jeden Fall schon mal Open Source.

3.1 Apache

Der wohl populärste aller Webserver steht nicht nur allen Linux – Distributionen zur Verfügung, sondern liegt auch als Win32 – Version vor. Die Installation ist einfach und nach der Grundkonfiguration unterstützt der Server SSI und CGI. SSI (**S**erver **S**ide **I**nclude) bietet die Möglichkeit ohne komplizierte Programmierung schnell ein HTML – Dokument mit dynamischen Inhalten auszurüsten. CGI-Programme (**C**ommon **G**ateway **I**nterface) können in vielen Programmiersprachen geschrieben sein und bekommen vom Server eine Laufzeitumgebung zur Verfügung gestellt (Umgebungsvariablen). Der Ausgabekanal ist meist stdout, der mit dem Webserver verknüpft ist. Ein Nachteil der CGI-Ausführung ist ihre relativ geringe Geschwindigkeit, da für jeden CGI-Aufruf eine neue Programm-Instanz (Prozess) ausgeführt wird. Und letzlich könnten auch Skripte von Fremden erstellt werden, die der Server ausführen würde. Das ist aus Sicherheitsgründen höchst problematisch. Um PHP ausführen zu können sind beim Apache Modifikationen der INI – Datei nötig. Ein Pluspunkt ist die sehr ausführliche Dokumentation, die auch in deutscher Sprache vorhanden ist.

Apache unterstützt Multithreading und Multiprotokolle, Ipv6, Bibliotheken für reguläre Ausdrücke, stellt Verschlüsselungen wie SSL zur Verfügung und vieles mehr. All diese Funktionalität stellt in der betrachteten Umgebung aber eher eine Belastung dar. Des weiteren ist er mit 4,24 MByte relativ groß.

Fazit: Apache ist der am meisten eingesetzte Webserver im Internet. Für die hier geforderten Leistungsmerkmale ist er eher ungeeignet, wie die Eigenschaften von CGI zeigen. Außerdem benötigt er für seine Arbeit ein perfekt angepasstes Betriebssystem, was im Embedded – und Echtzeitbereich wohl nur in den seltensten Fällen zur Verfügung steht. Er ist deswegen nur mit hohem Aufwand zu portieren.

3.2 Abyss Web Server X1

Der Abyss Web Server X1 ist so ressourcenschonend und kompakt, dass er sich mit einem 386er ab 33 MHz und 4 Megabyte RAM begnügt. Wer mehr Rechenpower als das Minimalsystem zur Verfügung hat, kann Skripte in den Sprachen PHP, Perl, ASP und Python in CGI-Skripten nutzen, ***falls die nötigen Interpreter installiert sind***. Außerdem lassen sich Fehlerseiten anpassen, Benutzer leicht verwalten, und SSI wird unterstützt. Der Hersteller bietet deutsche Sprachdateien auf seiner Website an.

Fazit: Nutzer, die nur ab und zu eine Webseite lokal testen möchten, sind mit Abyss sehr gut bedient. Mit 344 kByte ist er sehr schlank. Der Server ist schnell und bietet trotz der geringen Größe viel Funktionalität, bietet aber keinen Zugriff auf externe Prozesse. Das könnte zwar wieder mit CGI – Anwendungen erreicht werden, hätte dann aber wieder dieselben Nachteile wie beim Apache.

3.3 Xerver

Der kostenlose Java-Webserver Xerver unterstützt CGI-Skripts auf dem Rechner und beinhaltet selbst einen FTP-Server. Ein Wizard führt durch die Konfiguration im Browser, Standardfälle werden schnell abgehandelt. Bei Bedarf steht ein Interface für die Remote-Administration zur Verfügung, in der Default-Installation ist dieses aber nicht aktiviert. Applets können dynamische Inhalte in begrenzter Form möglich machen. Dies sind kleine Programme, die im Rahmen eines anderen Programms (Webserver) laufen. Ein Applet verfügt nur über eine begrenzte Anzahl von Funktionen, die keine eigenständige Funktionalität bieten. Daher auch der Name Applet (Mini – Applikation). Somit ist der Zugriff auf andere Prozesse versperrt

Fazit: Xerver ist zwar nicht so umfangreich wie beispielsweise Apache, ist aber dadurch auch einfacher zu handhaben. Ein praktischer Wizard hilft bei der Bedienung. Mit knapp über einem Mbyte ist aber auch er schon zu groß. Außerdem benötigt er als Systemvoraussetzung JAVA, was im Echtzeit – Umfeld eher unwahrscheinlich ist. Bietet keinen Zugriff auf externe Prozesse.

3.4 mini_httpd

Der Mini-Httpd ist ein kleiner Webserver und wurde von Jef Poskanzer entwickelt. Seine Leistungsfähigkeit ist nicht sehr groß, aber für kleinen und mittleren Datenverkehr völlig ausreichend. Er besitzt die Grundfunktionalität eines Webservers z.B.

- die Methoden Get, Post und Head
- Er ist CGI-fähig
- Authentifizierung
- die gängigen MIME-Typen
- Verzeichnislisten

Warum der Mini-Httpd überhaupt entwickelt wurde hatte zwei Gründe. Zum einen wollte man sehen, wie schnell ein altmodisch forkender Webserver auf den heutigen modernen Betriebssystemen ist. Er erreichte überraschend 90% der Geschwindigkeit eines Apache, obwohl dieser mit einem Thread-Pool arbeitet und dadurch die Prozesserzeugung überflüssig wird. Ein zweiter Grund war, man benötigte eine einfache Plattform um neue Webtechnologien schnell ausprobieren zu können (z.B SSL). In mein Projekt ist er, wegen seines monolitischen Aufbaus, nur schwer zu integrieren. Alle Systemfunktionen sind reine Linux-Funktionen. Kommunikation wird mit read() und write() realisiert und nicht systemübergreifend mit send() und recv(). Und die Sprache C sollte eigentlich schon durch die Sprache C++ ersetzt werden. Der Aufwand, den Server auf den von mir benötigten Stand zu bringen, wäre unverhältnismäßig. Aber als inspirierende Studie eignet er sich hervorragend.

3.5 thttpd

Auch der thttpd[1] ist ein Open Source Webserver, der sich durch seine Einfachheit und seine leichte Portierbarkeit auf andere Systeme auszeichnet. Als Besonderheit bietet er eine Bandbreitendrosselung, die in keinem anderen Webserver zu finden ist.

Diese Funktion ermöglicht dem Administrator, Dateitypen eine bestimmte Bitrate zuzuordnen. So ist es beispielsweise möglich, große Video-Dateien mit z.B. 20kbyte/s übertragen zulassen. Somit kann verhindert werden, dass der Server vor lauter Arbeit, nicht mehr auf andere Anfragen reagieren kann, und dadurch beim Clienten ständig das Timeout abläuft und die Warnung kommt "Server nicht erreichbar". Die Sicherheit wird erhöht, der Server wird aber künstlich verlangsamt. In zeitgesteuerten Prozessen von Echtzeitsystemen werden die Prozesse periodisch, zu einem genau definierten Zeitpunkt, zur Ausführung gebracht. Diese haben dann höchste Priorität. Es muss also nicht darauf geachtet werden, dass den Echtzeitprozessen genügend Zeit zur Verfügung steht. Im Gegenteil; diese Prozesse werden den Server einbremsen.

Fazit: Interessanter Server, aber er besitzt die selben Schwächen wie der Mini-Httpd und eine Schnittstelle, für den Zugriff auf nicht verwannte Prozesse, bietet auch er nicht.

Im Prinzip sind eigentlich alle vorhandenen Server eher für das Surfen im Internet gemacht und weniger für das Beobachten von zeitkritischen Prozessen. Eine andere Möglichkeit ist, eine erstellte Anwendung mit einer Kommunikationsschnittstelle auszustatten. Man würde dazu eine Bibliothek benötigen, die die gesamte Serverfunktionalität in Einzelkomponenten zur Verfügung stellt.

3.6 Die Bibliothek libhttpd

Die libhttpd kann verwendet werden, um grundlegende Webserverfähigkeiten in eine Anwendung einzubetten. Die Bibliothek bearbeitet sowohl statische als auch dynamisch generierte Inhalte, benötigt nur geringen Aufwand bei der Einbettung, und bietet viel Funktionalität, für Anwendungen, die eine webbasierte Schnittstelle benötigen. Sie ist in der aktuellsten Version 1.4 seit 2005 erhältlich. Das komplette libhttpd Handbuch ist online als PDF – Dokument [6] verfügbar. Sie hat aber auch einige Nachteile, die sich auf unser Projekt störend auswirken.

- keine dynamische Socketverwaltung
- keine Parallelität (serielle Anfrageabarbeitung)

1 Das erste t steht übrigens für Turbo

- Weicht der zu sendende Header vom Standard – Header ab, muss selbst einer geschrieben werden
- Sehr viele verwendete symbolische Konstanten und Makros tauchen in der Dokumentation nicht auf.

  Fazit : eigentlich eine geniale Sache, wenn da nicht die fehlende Parallelität wäre. Daher eher weniger geeignet.

Fazit : eigentlich eine geniale Sache, wenn da nicht die fehlende Parallelität wäre. Daher eher weniger geeignet.

Es wird sich also nicht vermeiden lassen einen eigenen Server zu entwickeln, der die von mir selbst geforderten Kriterien erfüllt. Eine zielorientierte Analyse soll helfen, Vorgehensweise und Techniken zu definieren.

4 Analyse

Der Server soll in erster Linie für das industrielle Umfeld entwickelt werden. Dort sind Schnelligkeit, Speichereffizienz, Erweiterbarkeit und leichte Wartbarkeit gefragt. Aus diesem Grund werde ich mich für die Sprache C++ entscheiden, die alle diese Eigenschaften besitzt. Sie besitzt alle Vorzüge von C, wurde aber um das Konzept der Objektorientierung und des Exception – Handlings erweitert. Dadurch ist auch sie, genau wie C, systemnah und damit schnell und durch das Klassenkonzept werden die erstellten Programme leichter erweiterbar, übersichtlicher und damit besser wartbar. Um die dynamische Speicherverwaltung muss man sich zwar selbst kümmern, aber gerade das ist für ein vorhersagbares zeitliches Verhalten der Serveranwendungen entscheidend. Alle betrachteten Server hatten den Nachteil, dass dynamisches Verhalten mit CGI – Fähigkeit oder serverseitig ausgeführten Scripten realisiert wird. Das hat jedesmal zur Folge, dass ein neuer Prozess gestartet werden musste. Und nicht verwannte Prozesse waren trotzdem nicht zu erreichen. Durch das Klassenkonzept kommt man zwar um das Erstellen von Anwendungen nicht herum, aber diese können direkt als Memberfunktionen aufgerufen werden ohne einen neuen Prozess erzeugen und starten zu müssen. In diese Memberfunktionen können jetzt die Routinen der Interprozesskommunikation eingebettet werden, die den Zugriff auf externe Prozesse erlauben. Die Anwendungen können also Systemfunktionalität nutzen, während die vorgestellten Server immer nur ihre Funktionalität zur Verfügung stellen konnten. In den bestehenden Server – Code muss nicht eingegriffen werden. Lediglich eine zu erstellende Anwendungs – Factory muss um die neue Anwendung erweitert werden. Das Klassenkonzept bietet aber noch mehr Vorteile. So können sämtliche Systemfunktionen in einer eigenen Klasse gekapselt werden und damit das Transferieren des Servers auf andere Plattformen, erheblich erleichtert werden. Ein Nachteil ist, dass nach so einer Erweiterung der Server neu kompiliert werden muss. Das setzt natürlich auch voraus, dass für die Zielplattform ein C++ Compiler existiert. Die Vorteile, die dieses Konzept aber im Gegenzug bietet, lassen diese Nachteile gegen Null tendieren.

Auf der Clientseite soll die Funktionalität eines normalen Webbrowsers ausreichen, um die Daten, die der Server liefert, zu visualisieren. Es werden in diesem Projekt Java-Scripte verwendet, die jeder Browser ausführen kann. Für kontinuierliche Anfragen an den Server werden Ajax-Routinen verwendet. Wenn sich Inhalte innerhalb einer HTML – Seite verändern, ist in der Regel ein Refresh (Seite neu laden) nötig. Mit Ajax aber können Aktualisierungen innerhalb einer Webseite vorgenommen werden, ohne das die Seite neu geladen werden muss. Somit beschränkt sich der Server auf das Liefern der Echtzeit-Daten, während die gesamte Arbeit zur Darstellung von Daten, auf die Clientseite ausgelagert wird. Zudem können heutige Browser standardmäßig eine Menge Datenformate interpretieren (z.B. txt, html, xml usw.).

Somit ist der Server von allen Aufgaben befreit, die über die Datenbeschaffung hinaus gehen und erfüllt dadurch die Kriterien Schnelligkeit und Speichereffizienz, und je weniger Funktionalität er bieten muss, umso leichter wartbar und plattformunabhängiger wird er. Aus diesem Grund wird auch die Anzahl von Clienten erstmal nicht beschränkt, zumal es sich hier nicht um einen Server handelt, der der breiten Öffentlichkeit interessante Informationen liefern könnte.

Die Abbildung 3 verdeutlicht noch einmal, wie das Zusammenspiel aller Komponenten aussehen soll.

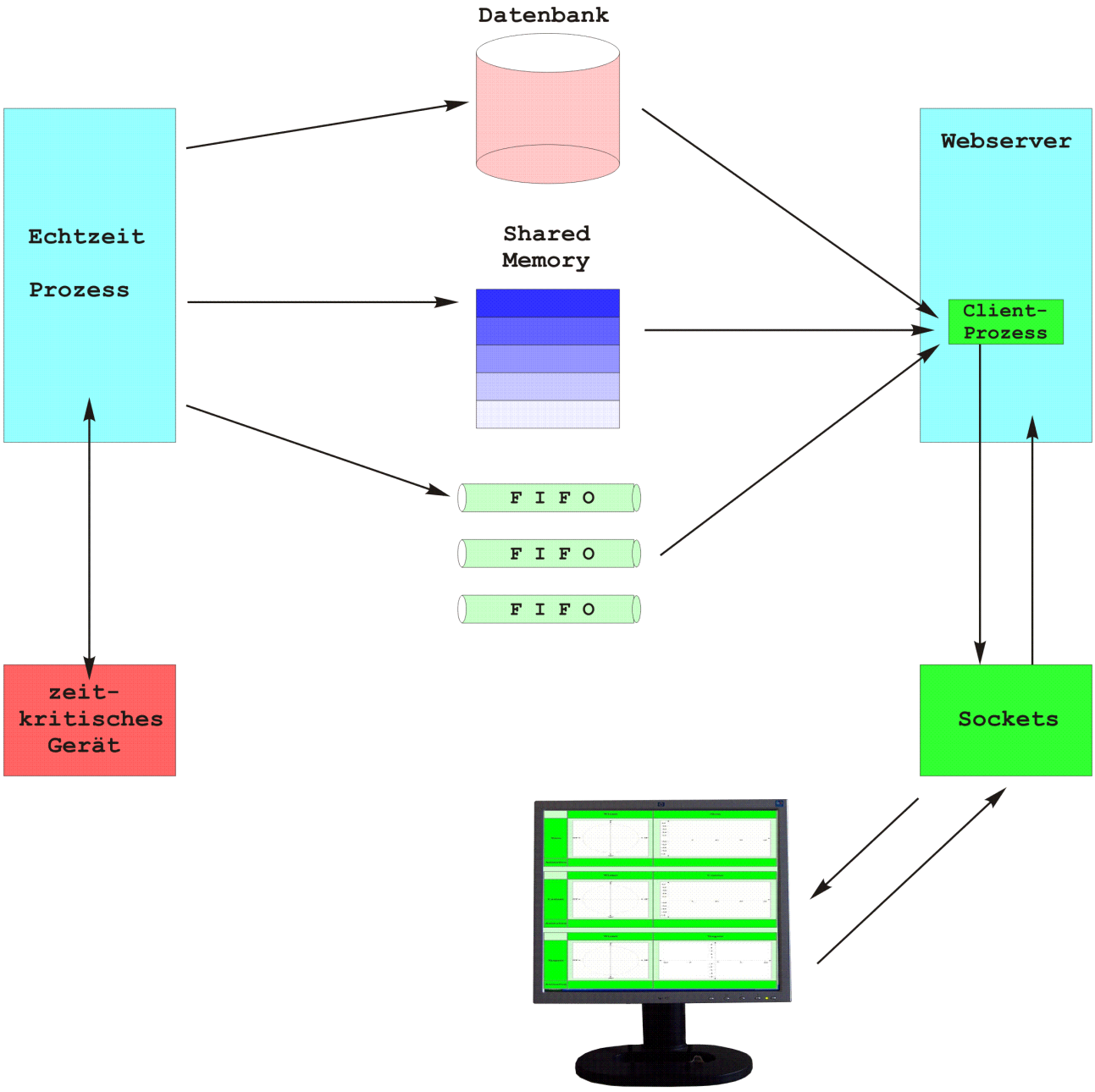


Abbildung 3: Schema für die Realisierung der Prozessvisualisierung

Auf welche Art die Daten zur Verfügung gestellt werden, ist vom vorhandenen System abhängig. Beispielhaft werden von mir FIFO's verwendet. Es muss erwähnt werden, dass zum Zeitpunkt dieser Arbeit kein externes Gerät zur Verfügung stand, so dass der Echtzeitprozess, die zu erfragenden Daten, simulieren muss. Es ist nicht wichtig, was für Daten der Prozess liefert, sondern in welchem Format sie geliefert werden, damit der Browser sie interpretieren kann.

Um dynamische Grafiken zu erstellen werde ich Integerwerte verwenden, die als Pixelkoordi – naten dienen. Für textuelle Nachrichten werden C-Strings übertragen, die der Browser als ASCII – Zeichen interpretiert. Auf eine Verschlüsselung der Daten wird hier verzichtet, da die Zahlen ansich keine Aussagekraft besitzen. Sie stellen ja nur Koordinatenpunkte dar, die erst in Verbindung mit der HTML – Seite ihre Bedeutung gewinnen.

5 Server erstellen

Für das Design des Servers sollten drei Eigenschaften zur Bedingung gemacht werden.
Er sollte :

- schnell sein und schnell bedeutet immer hardwarenah
- möglichst einfach und dadurch leicht wartbar sein
- erweiterbar sein und mit möglichst wenig Aufwand auf viele Plattformen portierbar sein (auch auf Echtzeit – Betriebssystemen).

Die Voraussetzungen die das Betriebssystem erfüllen muss sind minimal. Es muss netzwerkfähig sein (Socketfunktionen) und es muss eine Threadbibliothek unterstützen (in meinem Projekt werden Posix – Threads verwendet). Eine Programmiersprache , die die oben aufgeführten Anforderungen erfüllt ist z.B. C++. Sie baut auf der Sprache C auf und ist dadurch hardwarenah und schnell. C++ unterstützt weiterhin ein objektorientierte Klassenmodell, wodurch Applikationen modular und übersichtlich aufgebaut werden können. Und C bzw. C++ - Compiler sind für alle namhaften Prozessortypen verfügbar. Dadurch ist auch weitgehende Plattformunabhängigkeit gewährleistet. Als erstes soll eine Klasse, die den Datentransfer realisiert, erstellt werden. Zwei der am meisten verwendeten Protokolle heißen TCP (*Transmission Control Protocol*), eine verbindungsorientierte Übertragung und UDP (*Universal Datagramm Protocol*), ein verbindungsloses Protokoll. Die üblichen Webbrowser benötigen allerdings einen Verbindungsaufbau. Dennoch werden alle erstellten Netzwerkfunktionen als `virtual` deklariert, um den Server auch für andere Umgebungen nutzbar zu halten.

Bei TCP werden alle Pakete mit einer Sequenznummer versehen und damit wird sichergestellt, das alle Pakete beim Empfänger ankommen. Somit muss man sich nicht mehr um die Fehlerbehandlung bei Datenverlust kümmern.

5.1 TCP – Sockets

In diesem Abschnitt wird eine Socket – Klasse entworfen die sämtliche Systemfunktionen kapselt. Dazu werden Wrapper entworfen, denen alle das Namenskürzel 'oj_' als Kenn – zeichnung vorangestellt ist. Diese Angewohnheit aus der C – Zeit ist bei objektorientierten Sprachen eigentlich nicht mehr üblich, denn hier können Funktionen einer bestehende Klasse einfach überschrieben werden. Die Unterscheidung erfolgt dann an hand der Klassenzuge – hörigkeit. Dennoch kann es auch heute noch ganz sinnvoll sein um System – und Bibliotheksfunktionen von eigenen Implementierungen besser unterscheiden zu können und vor allem Namenskollisionen im selben Namespace zu verhindern ohne ständig die Klassenzugehörigkeit feststellen zu müssen. Alle erstellten Serverklassen werden hier nur als Deklaration aufgeführt und besprochen. Der komplette Quellcode ist auf der CD kompilier – fertig vorhanden.

```
class TCP_Socket
	{
	private:
		const int y;
		int sock, sock_client, ret;
		socklen_t server_len, client_len;
		struct sockaddr_in server_addr, client_addr;
	protected:
	public:
		TCP_Socket();
		virtual ~TCP_Socket();

		virtual int oj_socket();
		virtual int oj_bind();
		virtual int oj_listen();
		virtual int oj_accept();
		virtual void oj_options();
		virtual void oj_close( int socket);
	};
```

Wie schon erwähnt werden in den Wrapper – Funktionen die Systembefehle gekapselt. Diese können nun je nach zugrundeliegenden Betriebssystem ausgetauscht werden. Der objektorientierte und damit bessere Weg wäre, von der bestehenden Basisklasse abzuleiten. Darauf wird zu diesem Zeitpunkt erstmal verzichtet, da wie bereits erwähnt, Webbrowser das TCP – Protokoll benötigen. Außerdem müssten dazu alle zukünftigen Funktionen der Basisklasse bekannt sein, um den Polymorphismus voll nutzen zu können. Man kann sich nämlich bei dieser Technik leicht aussperren und kommt nur mit mühevollen Cast – Operationen wieder zurück ins Programm. Die TCP – Klasse bietet im Prinzip alle Funktionen zur Netzwerkkommunikation und ist daher als Basisklasse recht gut geeignet. Eine Erweiterung z.B. um das UDP – Protokoll könnte dann wie folgt aussehen.

```
class UDP_Socket : public TCP_Socket
	{
	private:
	protected:
	public:
		UDP_Socket();
		virtual ~UDP_Socket();

		virtual int oj_socket();
		virtual int oj_bind();
		virtual void oj_options();
	};
```

Die Funktionen zum Verbindungsaufbau (listen(), connect(), accept()) werden hier natürlich nicht benötigt, da UDP ein verbindungsloses Protokoll ist. Was allen Socket – Klassen gemein ist, ist die Socket – Verwaltung. Auch hier gibt es wieder mehrere Möglichkeiten. Man könnte sie den Funktionen send(...) und recv(...) überlassen. Diese Lösung ist die einfachste und funktioniert, hätte aber einen gravierenden Nachteil. Die Funktion recv(...) ist eine blockierende Funktionen, das heißt in diesem Fall, sie wartet so lange auf einen Socket, bis eine Eingabe[2] gelesen wird.

2 ein beliebter Punkt um Server zu blockieren, man eröffnet eine Verbindung ohne Daten zu schicken und blockiert somit einen Socket

Man könnte jetzt auf die Idee kommen recv(...) auf NONBLOCKED zu setzen und jeden Socket einzeln auf Daten zu überprüfen. Das würde aber einem Socket – Polling gleich kommen, was in der Regel den Prozessor stark belastet. Diese Lösung ist also für unser Anwendungsgebiet nicht akzeptabel.
Eine andere, weitaus komfortablere Möglichkeit Multiplexing zu realisieren, ist die Funktion select(...). Sie kann eine bestimmten Anzahl von Filedeskriptoren beobachten, ob sich ihr Zustand ändert. Jeder Client, der sich anmeldet, wird dieser Menge hinzugefügt und nach getaner Arbeit wieder entfernt. Die genaue Parameterstruktur und der Anwendungskontext kann im gut dokumentierten Quellcode nachvollzogen werden. Hier die zugehörige Klasse :

```
class SocketVerwaltung
      {
      private:
            fd_set fdlist, testfd;
            int result, rlen;

      protected:
      public:
            SocketVerwaltung();
            virtual ~SocketVerwaltung();

            virtual void fd_init( int socket);
            virtual void fd_copy();
            virtual int oj_select();
            virtual int oj_ioctl( int socket);
      };
```

Die Funktion, mit der das entfernen der Filedeskriptoren aus der aktuellen Menge realisiert wird, ist ioctl(...). Sie ist eine Universalfunktion und kann Informationen über alle möglichen Geräte liefern. Man kann mit ihr abfragen, ob eine CD im Laufwerk liegt und welcher Titel gerade läuft, wieviele Festplatten existieren, ob ein Drucker angeschlossen ist und ob er gerade ein Dokument druckt, oder eben auf einem bestimmten Deskriptor Daten zum Lesen vorhanden sind. Ob auf einem bestimmten Socket Daten gelesen werden können, erfährt man bei ioctl(...) durch den dritten Parameter. Dieser liefert die Anzahl der **noch** zu lesenden Bytes. Wird hier NULL gelesen, kann der Filedeskriptor aus der FD – Liste entfernt werden. Das könnte auch mit dem Timeout – Parameter von select() erfolgen. Mit ihm wird aber nur das Blockieren verhindert. Eleganter finde ich ioctl(...). Auf meinem System sind über 80 Makros implementiert, über die ioctl(...) Informationen liefern kann (siehe Man – Pages) und die ist längst nicht vollständig.

```
$ man ioctl_list
```

Dieser Server verwendet als zweiten Parameter das Makro FIONREAD und liefert die Anzahl der zu lesenden Bytes im dritten Parameter. Wird Null gelesen (es liegen keine Daten an) wird der Socket freigegeben.

5.2 Das HTTP – Protokoll

Das Protokoll, was auf TCP aufsetzt und welches hier zum Einsatz kommt, ist das Hyper Text Transfer Protokoll (HTTP). Es ist das Protokoll, mit dem sich Browser und Webserver Daten austauschen. Den Header, den der hier benutzte Browser zum Server schickt, habe ich mir mal ausgeben lassen.

```
GET  /Testseite.html  HTTP/1.1
Host:             localhost:8080
User-Agent:       Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.7)
                  Gecko/20050414 Firefox/1.0.3 SUSE/1.0.3-1.2
Accept:           image/png,*/*;q=0.5
Accept-Language:  de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding:  gzip,deflate
Accept-Charset:   ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive:       300
Connection:       keep-alive
```

Welche Informationen sind eigentlich im Rahmen der Aufgabenstellungen für uns interessant und wichtig ? Eigentlich nur die ersten beiden Zeilen. Ein wichtiges Detail ist noch die Angabe des Keep Alive. Beim HTTP / 1.0 waren alle Verbindungen nicht persistent. D.h., wenn ein Auftrag aus Sicht des Servers abgearbeitet war (z.B. angeforderte Seite wurde geliefert), wurde die Verbindung geschlossen. Wenn ein Client also eine persistente Verbindung aufbauen wollte, musste er explizit Keep Alive im Header mitliefern. In HTTP / 1.1 sind alle Verbindungen persistent, es sei denn, der Client schickt im Header 'close' mit. Der hier erstellte Server wird das Protokoll 1.1 unterstützen und somit das Schließen der Verbindungen selbst übernehmen. Er wird für jede Anfrage einen eigenen Prozess erzeugen der am Ende selbstständig alle Ressourcen wieder freigibt (siehe Socketverwaltung).

5.3 Kommunikation

Wie muss jetzt eigentlich der Header aussehen, den der Server schicken muss, damit die Daten, die er liefern soll, richtig interpretiert werden ?

```
HTTP/1.1 200 OK
Date              : Fri 15 Jun 2007 16:25:17 GMT
Server                 : Bachelor-Server/1.0
Last-Modified     : Fri 15 Jun 2007 13:47:12 GMT
Content-Length    : 1234
Connection        : close
Content-Type      : text/html
```

Wichtig ist das verwendete Protokoll, die Anzahl der Bytes, für die der Client Speicher bereit stellen muss und der Typ der Daten, die gesendet werden.
Von besonderer Bedeutung für diesen Server ist der Zeitstempel. Browser haben die Eigenschaft, Daten gleichen Namens als identische Daten zu interpretieren und somit alte Daten, aus Performancegründen, aus dem Cache zu holen. In dem der Zeitstempel mitgeliefert wird, werden alle Daten als neu erkannt, obwohl alle anderen Informationen des Headers identisch sind.
Damit der Server auch andere Protokolle unterstützen kann, wird von einer Basisklasse abgeleitet. Dadurch könnte der Server unterschiedliche Clienten mit unterschiedlichen Protokollen bedienen.

Als erstes die Basisklasse :

```
class SysCom
      {
      private:
            char *temp;
            int i, laenge, temp_laenge, ngesamt;

      protected:
            char message[255];
      public:
            SysCom();
            virtual ~SysCom();
            virtual int sendblock( int socket, char *puffer,     int
groesse);
            virtual int sendpuffer( int socket, char *puffer);
            virtual int readdaten( int socket, int fp);
      };
```

Mit der Funktion sendblock(...) können Datenpakete immer gleicher Größe verschickt werden (z.B. Dateien), während sendpuffer(...) variable Datenmengen überträgt. Die Größen – erkennung erfolgt hier durch den Terminator.
Diese Funktionen sind für alle Protokollklassen identisch, nur der Header ist im Einzelfall verschieden. Hier jetzt die Headerklasse :

```
class HTTP_Header : public SysCom
      {
      private:
            struct tm *ptm, *pftm;
            time_t stime;
      protected:
      public:
            HTTP_Header();
            virtual ~HTTP_Header();

            virtual void send_HTTP_Header
                  ( int socket, int code, const char *phase,
                    int laenge, time_t *pftime);
      };
```

Diese Headerklasse besteht, wie zu sehen, nur aus einer Funktion : send_HTTP_Header(...) mit den oben besprochenen Headerinformationen als Übergabeparameter. Wie die genaue Implementierung aussieht, ist wieder dem Quellcode im Anhang zu entnehmen.

5.4 Jede Anfrage ein Prozess

Nachdem alle Einzelkomponenten für die Client / Server – Kommunikation erstellt sind, kann die Event – Loop zur Auftragsabarbeitung entwickelt werden. Sie befindet sich im Hauptprogramm und wird für jede Anfrage einen eigenen Prozess erzeugen. Man kann diese Aufgabe mit Systemprozessen oder mit Threads realisieren. Die Unterschiede liegen in der Art der Kommunikation und im Adressraum [7].
Damit Prozesse untereinander Daten austauschen können, müssen sie untereinander synchronisiert werden. Desweiteren entsteht bei Prozessen ein enormer Aufwand beim Duplizieren des Namensraumes.

Beides wird mit Threads vermieden. Sie besitzen einen gemeinsamen Adressraum und benutzen somit dasselbe Codesegment, Datensegment, Heap und alle anderen Zustandsdaten. Sie haben also alles, was auch Prozesse besitzen, nur der Austausch von Daten und die Kommunikation untereinander wird erheblich erleichtert. Da sie aber denselben Adressraum benutzen bedeutet dies; stürzt ein Thread ab, stürzen alle anderen Threads mit ab.
Ein weiterer Unterschied besteht darin, dass Threads unabhängige Befehlsfolgen innerhalb eines Prozesses darstellen. Sie sind also Bestandteil des Serverprogramms und können nur innerhalb dieses Programms arbeiten. Da sich also Threads leichter handhaben lassen werde ich sie auch einsetzen. Verwenden werde ich Posix – Threads, da diese Bibliothek auf den meisten Plattformen verfügbar ist. Um die Funktionalität zu gewährleisten sind nur zwei Threadbefehle nötig.

```
pthread_detach( pthread_self());
pthread_create( &pt, NULL, &execute, (void *)new_client);
```

Und hier der Code des restlichen main – Programms :

```
int main( int argc, char **argv)
{
   int new_client = 0;
   pthread_t pt;
   TCP_Socket *tcp_sock;

   tcp_sock = new TCP_Socket();
   tcp_sock->oj_socket();
   tcp_sock->oj_options();
   tcp_sock->oj_bind();
   tcp_sock->oj_listen();

   while( 1)
   {
      new_client = tcp_sock->oj_accept();
      pthread_create( &pt, NULL, &execute, (void *)new_client);
   }
   delete( tcp_sock);
   return 0;
}
```

Die Funktion execute(...) beinhaltet die Socketverwaltung und den HTTP_Deamon, wird also solange ausgeführt, bis alle Daten gelesen und vom Server beantwortet worden sind. Erst wenn sich keine Daten mehr auf den Socket befinden, bricht der Thread ab und beendet sich durch pthread_detach(...).
An dieser Stelle wurde das objektorientierte Design einmalig verlassen, da man sich mit den Threads in einem Dilemma befindet, was bei Systemprozessen nicht eingetreten wäre. Möchte man einen Thread aus einer Klasse heraus starten, d.h. **den Systembefehl in einer Memberfunktion kapseln**, würde die Struktur der Klasse in etwa folgendes Aussehen haben.

```
class Thread
   {
   private:
      pthread_t th;
   public:
      void thread_start();
      void *execute()         { /* auszufuehrender Code */ }
   };
```

```
void Thread::thread_start()
{
   pthread_create( &th, NULL, (Thread::execute), NULL);
}

int main( int argc, char **argv)
{
   Thread *pth;
   pht = new Thread();
   pth->thread_start();
        . . .
   return 0;
}
```

*execute() ist ein Zeiger auf eine C – Funktion. In C++ wird daraus jetzt ein Zeiger auf eine Memberfunktion der Klasse Thread. Damit das obige Codefragment funktioniert, muss der Zeiger vorher bekannt sein oder man deklariert ihn statisch. Bekannt ist der Zeiger nicht, da ja das Objekt erst zur Laufzeit erzeugt wird. Bleibt noch die statische Variante :

```
static void *execute()       { /* auszufuehrender Code */ }
```

Jetzt ist die Funktion zwar schon vor `main` bekannt, statische Funktionen haben aber eine Lebensdauer bis zum Programmende. Man kann also den Speicher, der mit new Thread() allokiert wurde, nicht vollständig mit delete(pth) wieder freigeben. Der Server wird also bei jeder Anfrage mehr Speicher allokieren als er freigibt und würde aus Speichermangel seinen Dienst irgendwann einstellen. Aus diesem Grund werde ich die execute – Funktion außerhalb der Klasse deklarieren und damit diesem Problem aus dem Weg gehen. Das objektorientierte Design des Servers wird an dieser Stelle aber etwas aufgeweicht.

5.4.1 Der HTTP – Deamon

90% der Threadfunktionalität erledigt die Klasse `HTTP_Deamon`, der für die Abarbeitung der Anfrage zuständig ist. Die restlichen 10% entfallen auf Übergabeparameter, Stringbearbeitung usw. Deamon deshalb, da man die Arbeit dieser Klasse selbst nicht sieht. Erst im Browser wird die Arbeit des Deamon für den User sichtbar. Nach der Deklaration der Klasse folgen dann einige Erklärungen zu den Besonderheiten.

```
class Http_Deamon
      {
      private:
            /*   ... Anlegen von Puffern fuer Header und Daten
                 siehe Quellcode ...   */
            struct stat fst;
            FILE *fd, *file;
            HTTP_Header header;
            QueryString *qs;
            Anwendung *anw;
      protected:
      public:
            Http_Deamon();
            virtual ~Http_Deamon();
            virtual int gettoken( char *puffer, int n,
                    char *zeichen, int zeichen_len, char delim);
            virtual int http_deamon( int socket);
            virtual int prog_factory( int socket, char *uri);
      };
```

Als erstes trifft immer ein Client – Request ein, d.h. der Client (Browser) baut eine Verbindung mit dem Server auf, bekommt einen Socket zugewiesen. Der Deamon liest den Socket aus und speichert die für ihn wichtigen Daten[3] zwischen. Um eine Anfrage auszuführen sind 3 Informationen nötig. Die Aufrufmethode, die Datei und der Pfad dorthin und das Protokoll, mit dem gesendet werden soll. Für die Aufgaben, die dieser Server erfüllen soll, ist es völlig ausreichend, dass er die Methoden[4] 'Get' und 'Post' unterstützt.
Danach muss die Datei und der Pfad zur Datei, aus dem Header herausextrahiert werden. Dazu wird mit gettoken() die gesamte erste Headerzeile nach bestimmten Zeichen durchsucht. Der String, der sich zwischen dem ersten '/' (Slash) und dem darauffolgenden Leerzeichen befindet, enthält den Weg zur angeforderten Datei. Der Rückgabewert ist die Länge des in einem Puffer gespeicherten Strings.

5.4.2 Get oder Post

Get bzw. Post stellen Request / Response – Methoden dar, die hier kurz behandelt werden sollen. Im Normalfall wird die Get – Methode als Default – Einstellung benutzt. Sie sagt aus, dass die übergebenen Daten hinter der URL angehängt werden. Bei der Post – Methode wird der Inhalt im Body – Blocks zurückgeliefert (siehe Abschnitt 5.3 `sendblock(...)`). In diesem Fall wurde entweder eine Datei angefordert oder es kommen Daten im reinen Textformat. Der Browser stellt dies im mitgelieferten Content – Type fest. Der Server erwartet bei Get und Post, dass die mitgelieferte Daten als Wertepaare vorliegen. Die Post – Methode unterscheidet sich in ihrer Funktionalität nicht von der Get – Methode. Da die Get – Methode ihre Daten an die URL anhängt, ist der Umfang der Daten, im Unterschied zu Post, begrenzt. Post wird meistens zur Übermittlung von Formulardaten verwendet. Es muss bei dieser Methode als Content – Type der Applikation – Wert mitgeteilt werden.

```
Content-Type      : application/x-www-form-urlencoded
```

Diese Headermitteilung ist bei Formulardaten zwingend vorgeschrieben da sonst eine erfolgreiche Datenübertragung nicht garantiert werden kann. Wie oben beschrieben, liefert der hier entwickelte Server, Daten von einem Echtzeitprozess in Byte – Form. Die Datenmenge ist daher relativ überschaubar. Deshalb ist es hier relativ egal, mit welcher Methode die Kommunikation abläuft. Implementiert ist allerdings, wie im Abschnitt Kommunikation zu sehen, die Übertragung eines Bodyblocks.
Bei großen und kommerziellen Servern sollte aber berücksichtigt werden, dass die mit Post gesendeten Daten nicht in Proxy – Logdateien oder im Referer auftauchen [8]. Für den Bereich der sensiblen Daten sollte dies hier noch erwähnt werden. Dieser Umstand spielt bei dem hier entwickelten Server keine Rolle.

5.4.3 Der normale Datei – Transfer

Ist die angeforderte Datei vorhanden, wird sie zum Lesen geöffnet. Mit der Funktion `stat` können Dateiattribute erfragt werden, unter anderem auch die Größe der Datei. In derselben Größe wird ein temporärer Speicher allokiert, in der die Datei kopiert wird. Dieser Speicher wird dann der Sendefunktion übergeben, die die Datei an den Clienten schickt. Der String wird aber noch nach anderen Zeichen durchsucht.

3 eigentlich die für mich wichtigen Daten, denn ich habe ihn ja programmiert
4 Was es mit diesen Methoden auf sich hat, wird im Kapitel 3, Erstellen der Clientseite, beschrieben

5.5 Parameterübergabe

Es wird überprüft, ob im String ein '?' (Fragezeichen) vorhanden ist. Befindet sich ein Fragezeichen in diesem String, wurden vom Clienten Parameter an den Server übermittelt.
Im Normalfall muss sich jeder Webentwickler selbst darum kümmern, ob und wie er sich die Parameter aus dem Enviroment besorgt. Dieser Server wird diese Aufgabe dem Anwendungs – Programmierer abnehmen. Die Prozessausführung wird dadurch erheblich beschleunigt.
Parameter können aus Eingabe – Formularen stammen oder sie werden der Anfrage direkt angehängt. Wie sie übergeben werden müssen ist genau definiert. Parameter beginnen immer nach dem ersten Auftreten eines Fragezeichens ('?') und bestehen aus zwei Werten; dem Namen des Parameters und dessen Wert[5].

?	Nach dem Fragezeichen beginnen die Parameter
&	Trennzeichen von mehreren Parametern
=	Trennung eines Parameterpaars in Name und Wert
+	Damit werden Leerzeichen gekennzeichnet
%xx	Damit werden Sonderzeichen dargestellt (xx steht beispielhaft für zwei Hex-Zahlen)

Beispiel :
/test.html?Name1=Oliver&Name2=J%FBnicke&Textfeld1=HAW+Hamburg
Die Klasse QueryString macht aus diesen etwas kryptisch aussehenden String wieder einen Klartext:

```
class QueryString
      {
      private:
            char *paare[MAX_PAARE];
      protected:
      public:
            QueryString();
            virtual ~QueryString();

            ListenEintrag *anfang;
            ListenEintrag *ende;
            int queue_put( ListenEintrag *le);
            ListenEintrag *queue_get();
            void ausgabe();

            void param_filter( char *uri, char *param);
            void hex_to_ascii( char *string);
            char convertiere( char *hex);
            char *strcopy( const char *string);
            virtual void print_error( char *string);
            virtual int paare_erstellen( char *string);
      };
```

Wie immer, sind auch hier nicht alle Hilfsvariablen aufgeführt. Die Klasse erzeugt aus Wertepaaren eine Queue von Parametern. Damit ist die Reihenfolge der Übergabeparameter identisch mit der Reihenfolge in der Queue. Diese Parameterliste kann jeder Anwendung übergeben werden. Auffällig ist wahrscheinlich die Funktion strcopy(...). Diese dupliziert einen String, damit nicht auf dem Originalstring gearbeitet werden muss.

5 Das HTTP – Protokoll ist reiner ASCII – Text. Werden also Zahlen übergeben, müssen sie vom Stringformat in die unterschiedlichen Zahlenformate umgewandelt werden

Ein Listeneintrag ist folgendermaßen aufgebaut :

```
class ListenEintrag
      {
      private:
            char *variable;
            char *wert;
      protected:
      public:
            ListenEintrag()                     { next = 0; }
            virtual ~ListenEintrag();
            ListenEintrag *next;

            char *get_variable()          { return variable; }
            int set_variable( char *var);
            char *get_wert()              { return wert; }
            int set_wert( char *w);
      };
```

Jedes Objekt der Klasse 'ListenEintrag' speichert ein Wertepaar an Parametern. Über die get – und set – Methoden kann auf die Einträge zugegriffen werden.

5.6 Anwendungen

Alle Anwendungen, mit denen der Client interagieren soll, haben dasselbe Schema. Sie sind von einer Basisklasse `Anwendung` abgeleitet und können so immer auf dieselbe Art angesprochen werden. Mit Anwendungen sind Programme gemeint, die mit Echtzeit – Prozessen kommunizieren können und somit Informationen über diesen Prozess an den Clienten liefern.
Exemplarisch für alle Anwendungen soll anhand eines Beispiels die grundsätzliche Vorgehensweise demonstriert werden. Das Beispiel soll eine Zeitanfrage des Clienten mit der aktuellen Systemzeit der Serverseite beantworten. Diese Anwendung kommuniziert noch nicht mit Echtzeit – Prozessen und soll nur den Aufbau einer Anwendung verdeutlichen.

```
class SystemZeit : public Anwendung
      {
      private:
            char daten[255];
            time_t zeit;
            HTTP_Header header;
      protected:
      public:
            SystemZeit();
            virtual ~SystemZeit();
            void oj_send_daten( int socket, QueryString *qs);
      };
```

Im Konstruktor wird die Systemzeit erfragt und in den Datenpuffer kopiert. Damit später Pakete gepackt werden können, die aus mehr Informationen bestehen als der Systemzeit, wird schon jetzt die Funktion `sprintf()` benutzt, mit der mehrere Daten mit unterschiedlichen Formaten in einen Puffer geschrieben werden können.

```
SystemZeit::SystemZeit() : Anwendung()    // Konstruktor
{
     sprintf( daten, "%s", ctime( &zeit));
}

void SystemZeit::oj_send_daten( int socket, QueryString *qs)
{
      header.send_HTTP_Header( socket, 200, "Aktuelle Systemzeit",
                               strlen( daten) + 1, NULL);
      header.sendblock( socket, daten, strlen( daten));
      header.sendpuffer( socket, "\r\n");
}
```

In der Sende – Funktion könnte natürlich noch viel mehr Code stehen, wichtig sind allein die drei Sendeschritte :

1. Header senden
2. Datenblock senden
3. Übertragung serverseitig beenden

Wie jetzt selbst entwickelte Anwendungen in den Server integriert werden müssen, ist Thema des nächsten Abschnitts.

5.7 Die Anwendungs – Factory

Eine Objekt – Factory ist ein Konstrukt, welches zur Laufzeit die unterschiedlichsten Objekte erzeugen kann. Bedingung ist nur; sie müssen alle von einer gemeinsamen Basisklasse abgeleitet sein. Der Basiszeiger `Anwendung *anw` ist in der Headerdatei des HTTP-Deamon bekannt gemacht (siehe Abschnitt 5.4.1).
Somit wird der Server zwar mit jeder Anwendung größer, der erzeugte Anfrage – Thread bleibt in seiner Größe allerdings unverändert.

```
if( !( strcmp( uri, "/uhrzeit")))
{
      anw = new SystemZeit();
      anw->oj_send_daten( socket, qs);
      delete( anw);
}
```

Es sind also 4 Schritte nötig um eine neue Anwendung zu integrieren :

1. Neue Funktion von Anwendung ableiten und entwickeln.
2. In der Headerdatei des HTTP-Deamon bekannt machen.
3. Zur Factory hinzufügen.
4. In den Makefile eintragen und neu kompilieren.

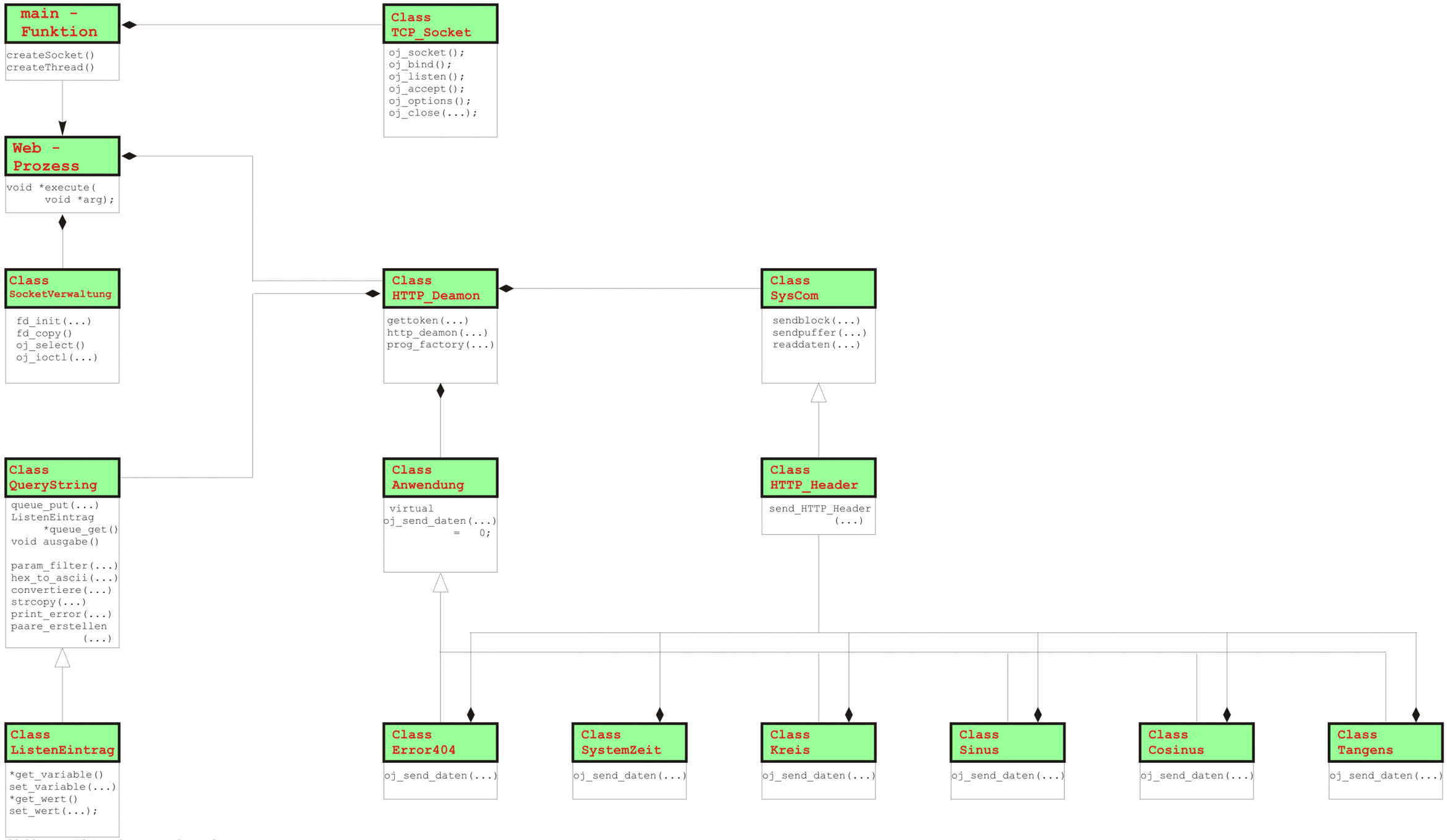
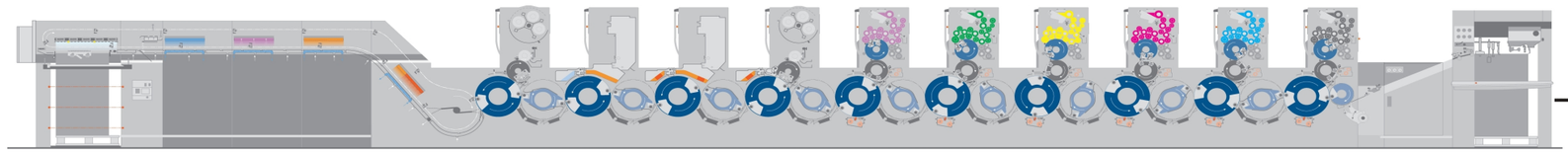


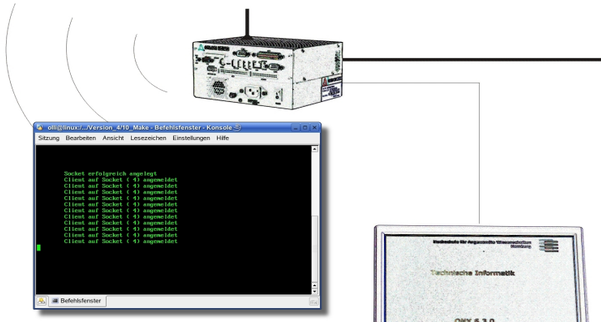
Abbildung 4: Klassendiagramm des Webservers

Abbildung : Heidelberger Druckmaschinen AG



Simulation der Prozessvisualisierung

Eine mögliches Anwendungs - Szenario zeigt dieses Bild.
Der entwickelte Webserver wurde auf dem Steuerungsrechner der Industrieanlage installiert und gestartet. Von seinem Home - PC oder über das Firmen - Netzwerk kann sich der Anlagenbetreuer mit der gültigen IP - Adresse einloggen. Er kann nun für jeden Anlagenteil die entsprechende Seite aufrufen und je nach Funktionalität der Seite, Parameter abfragen oder auch Steuerungsbefehle an die Maschine senden. Die abgebildet Druckmaschine dient nur der Darstellung. Entsprechende Frage - und Antwortdaten wurden von den erstellten Anwendungen simuliert. Sie können allerdings jeder Zeit gegen reale Sensordaten der Ma - schine ausgetauscht werden.



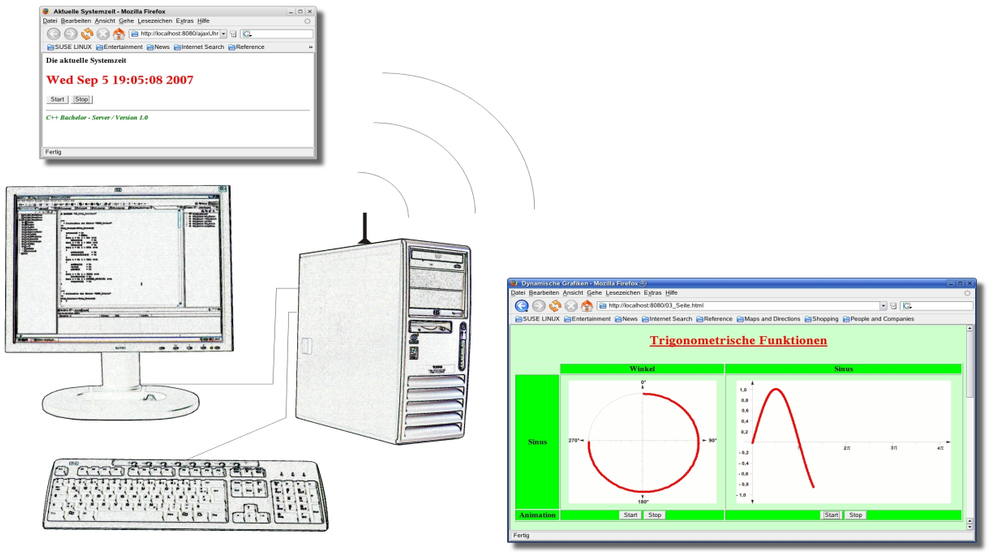


Abbildung 5: mögliches Anwendungsszenario des Webservers

Die gesamte Aufrufsemantik im HTTP – Deamon soll die folgenden Grafik noch einmal verdeutlichen :

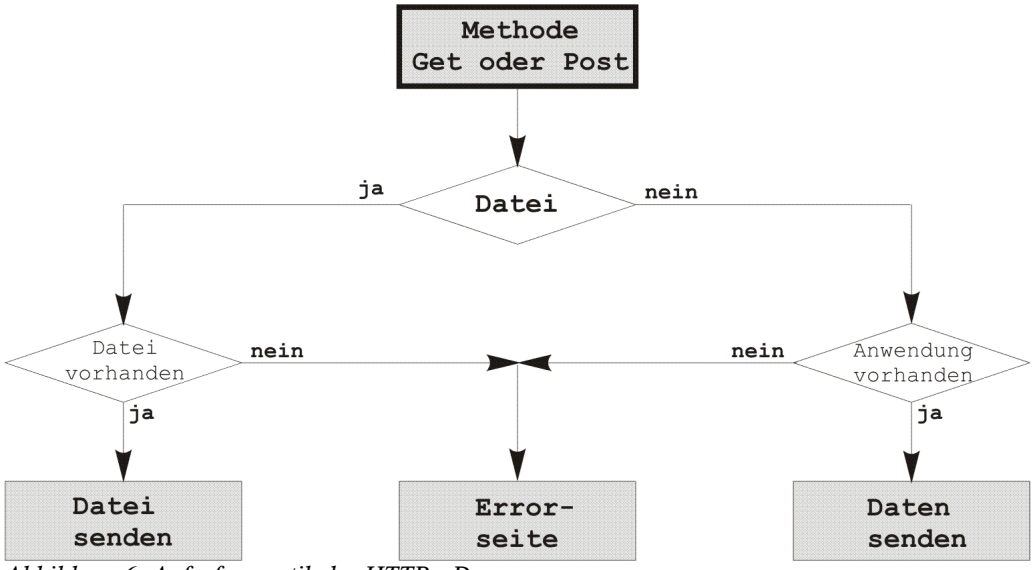


Abbildung 6: Aufrufsemantik des HTTP - Deamon

Derzeit ist die Factory und die Anwendungen Teil der Klasse HTTP_Deamon. Die Strukturierung könnte aber auch anders aussehen. Z.B. könnte eine eigene Factory - Klasse erstellt werden, die von den Klassen HTTP_Deamon und Anwendung erbt. Desweiteren wird natürlich mit jeder Anwendung das if / else – Konstrukt für die Suche nach Anwendungen immer länger. Um Suchwege zu optimieren, könnten Bäume oder Hashtabellen implementiert werden. Dieser Aspekt ist eine andere Geschichte und soll an dieser Stelle nicht weiter thematisiert werden.
Damit hat der Server sämtliche Funktionalität, die er benötigt, um seine Aufgabe zu erfüllen. Das gesamte Klassendiagramm ist in der nachfolgenden Grafik noch einmal vollständig abgebildet. Als nächstes soll die clientseitige Funktionalität erstellt werden.

6 Client erstellen

Um die Arbeit des Servers auf der Clientseite (Browser) zu beobachten, muss als erstes eine Testseite erstellt werden. Diese Seite soll die spätere Realität so nah wie möglich widerspiegeln. Dazu wird in diesem Beispiel ein kleiner roter Punkt verwendet, der als Gif – Datei hinterlegt wird. Der Server liefert jetzt Messdaten, die als Bildschirmkoordinaten interpretiert werden. Für jede dieser Koordinaten wird ein Objekt des roten Punktes erzeugt und auf dem Bildschirm sichtbar gemacht. Im Laufe der Zeit werden sich die einzelnen Punkte, wie bei einem Mosaik, zu einer komplexen Grafik entwickeln, die dann die Arbeit eines Prozesses über einen gewissen Zeitraum dokumentiert. Da keine reelle Anlage existiert, deren Daten aufgezeichnet werden können, werden vom Prozess Koordinaten erzeugt, die sich zu trigonometrischen Funktionen (Sinus, Kosinus, Tangens, Kreis) aufbauen. Alle Anwendungen sollen auf einer Seite beobachtet werden können und sie sollen an – bzw. abschaltbar sein. Um die Anzeigefenster an bestimmten Koordinaten positionieren zu können, gibt es auf Webseiten nur die Möglichkeit mit Tabellenrastern [9] bestimmte Bereiche zu reservieren. Eine Tabelle ist in HTML folgendermaßen aufgebaut[6] :

Eine Tabelle wird mit `<table> ... Tabelleninhalt ... </table>` angelegt. Das Tag – Paar `<tr>` und `</tr>` (table row) definiert eine neue Zeile und `<td>` bzw. `</td>` (table data) den Zelleninhalt. Die eingefügten Start – und Stopp – Buttons werden <input ... > angelegt. Größe und Farbe der Zellen können mit den Tags height, width und bgcolor festgelegt werden. Eine komplette Zelle der Testseite sieht dann wie folgt aus :

```
<table border>
<tr>
   <td width=380 height=20 bgcolor="#00FF00"><h3>
   <center>Winkel</center></h3></td>
   <td width=380 height=350 id="winkel_td_1">

   <img src="Grafiken/Kreis.gif" id="winkel_img_1">
   <input type="button" name="start" value="Start"
                        onClick="running=true; run()">
   <input type="button" name="stop" value="Stop"
                        onClick="running=false">
</tr>
</table>
```

Die Funktionen, die onClick innerhalb der Tabelle startet bzw. wieder stoppt, werden im Abschnitt 6.4.1 erstellt und beschrieben.

6.1 Hintergrundgrafiken

Die Hintergrundgrafiken einer Anwendungszelle können mit jedem Grafikeditor leicht erstellt werden (bei einfachen Grafiken), dann als .gif – Dateien abgespeichert und schließlich mit dem <img ... > - Tag in die HTML – Seite eingebunden werden. Zu beachten ist, dass die Größe der Grafik der Größe der Zelle entspricht, da sonst das gleichmäßige Raster der Zelle zerstört wird. Für die Testseite wurden Koordinatensysteme für die trigonometrische Funktionen angefertigt, die mit CorelDraw erstellt worden sind.

6 Hier soll kein Einstieg in HTML gegeben werden, sondern nur die verwendeten Techniken kurz dargelegt werden

6.2 Grundlagen von HTML und DOM

Um dynamische Grafiken in die oben erstellte Tabelle einzubinden, können zwei unterschiedliche Ansätze verfolgt werden. Zum einen kann der Scriptcode direkt in der HTML – Seite untergebracht werden, oder der Code wird in Skriptdateien ausgelagert. Die ausgelagerten Dateien müssen dann mit dem Script – Tag in die HTML – Seite eingebettet werden. Übersichtlicher ist die Datei – Variante und kommt auch hier zur Anwendung, aber letztlich liefern beide Ansätze dasselbe Ergebnis. Der Browser lädt beim Start der Seite den Inhalt aller Scripte und nicht die Script – Dateien. Namenskonflikte gelten also nicht nur innerhalb einer Script – Datei, sondern über alle Scripte, die von einer Seite geladen werden (beliebter Anfängerfehler, auch von mir). Damit auf bestimmte Objekte innerhalb einer HTML – Seite zugegriffen werden kann, ohne die gesamte Seite neu zu laden, müssen HTML – Seiten dem DOM – Standard entsprechen. Das W3C Document Object Model (DOM) ist eine plattform – und sprachunabhängige Schnittstelle für den Zugriff auf XML und XHTML Dokumente. Eine vollständig übertragene Webseite repräsentiert nach diesem Standard einen hierarchisch angeordneten Dokumentbaum. Das folgende Beispiel soll die Zusammenhänge bildlich darstellen.

```
<html>
   <head>
     <title>JavaScript und Dom</title>
   </head>
   <body>
     <h1>kein Koelner Dom</h1>
     <p>Ein<strong><em> kleiner </em></strong>hierarchischer Dom</p>
   </body>
</html>
```

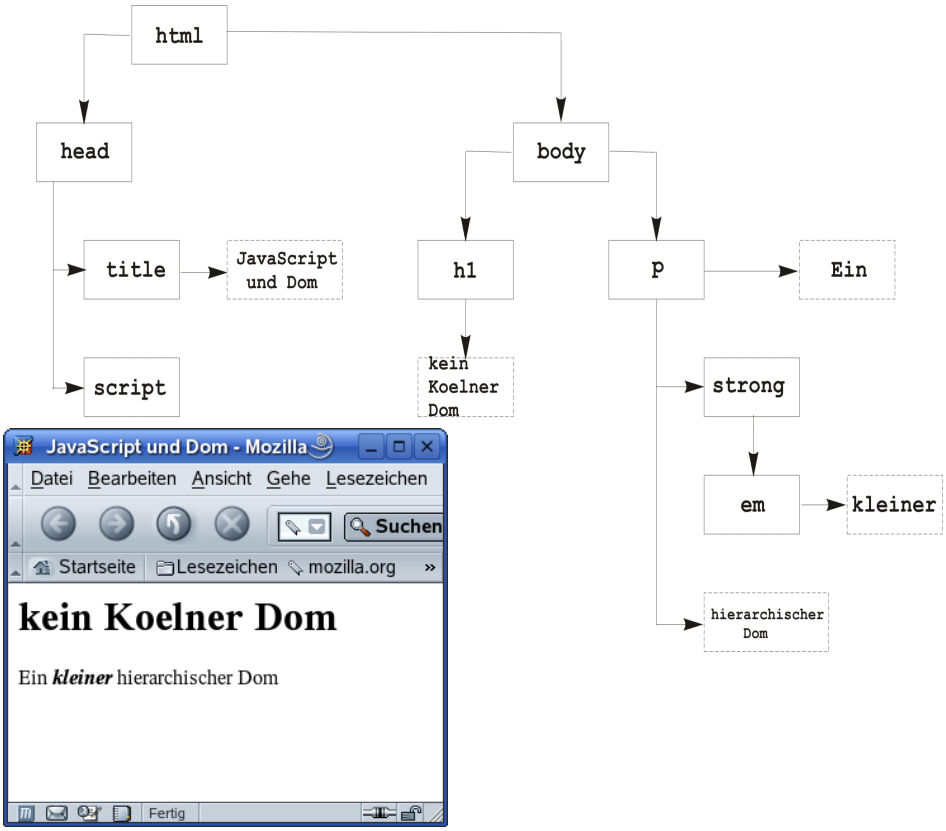


Abbildung 7: DOM - Hierarchie - Beispiel

Entlang dieses Baumes kann nun auf die Elemente zugegriffen werden. Will man z.B. den Textstyle verändern, muss man sich nun den Baum von oben nach unten entlang hangeln.

```
document.body.childNode[0]  ist das <h1>    Tag und
document.body.childNode[1]  ist das <p>    Tag
```

Möchte man nun das Wort “Ein“ auf dem Dokument verändern und z.B. durch das Wort “Hallo“ ersetzen, so sieht das folgendermaßen aus :

```
document.body.childNode[1].childNode[0] =   Hallo
```

Um als Programmierer also effektiv arbeiten zu können, muss man sein HTML – Formular immer im Hinterkopf haben und den Code gut einrücken um nicht den Überblick über den DOM zu verlieren. Da im Rahmen dieser Arbeit nicht auf alle Methoden und Attribute eingegangen werden kann, werde ich mich in den folgenden Ausführungen auf die Methoden beschränken, die Bestandteil der hier erarbeiteten Scripte sind.

6.3 Grundlagen von Ajax

Der Begriff **Ajax** steht für **A**synchron **J**avascript **A**nd **X**ml und beschreibt keine neuentwi – ckelten Webtechniken, sondern will bestehende Techniken in einer sinnvollen und kreativen Art und Weise neu beleben.
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. Die eigentliche Neuerung besteht in der Tatsache, dass nur gewisse Teile einer HTML – Seite oder auch reine Nutzdaten bei Bedarf nachgeladen werden können, womit Ajax eine Schlüsseltechnik zur Realisierung des Web 2.0 darstellt [10].
Die wesentlichen Techniken, die in Ajax zur Anwendung kommen, sind:

- XHTML[7] und CSS[8] für die Formaatierung einer Webseite
- XML[9] oder JSON[10] für den Datenaustausch
- XSLT[11] für die Transformation
- DOM für den dynamischen Zugriff auf den Dokumentenbaum
- Das `XMLHttpRequest` – Objekt für eine Client / Server – Kommunikation auf asynchroner Basis [8]
- JavaScript als Schnittstelle all dieser Komponenten.

Sie sind hier die vorherrschenden Themen. Es ist nämlich nicht möglich eine Anfrage an den Server zu stellen und danach für unbegrenzte Zeit, Daten in einem bestimmten Intervall zu empfangen. Der Client (Browser) kann jeder Anfrage nur eine Antwort zuordnen.
Alternative Techniken[12] können den entsprechenden Seitenangaben entnommen werden.

7 http://de.wikipedia.org/wiki/XHTML
8 http://de.wikipedia.org/wiki/Cascading_Style_Sheets
9 http://de.wikipedia.org/wiki/XML
10 http://de.wikipedia.org/wiki/JSON
11 http://de.wikipedia.org/wiki/XSLT
12 http://www.teialehrbuch.de/Kostenlose-Kurse/AJAX/20846-Alternativen-zu-AJAX.html

6.3.1 Ajax – Request / Response

Bei der Arbeit mit Ajax muss als erstes ein `XMLHttpRequest` – Objekt erzeugt werden. Und hier sind sich die Hersteller von Browsern zum ersten Mal nicht einig. Der Internet Explorer von Microsoft benötigt ein ActiveX – Objekt mit folgender Syntax :

```
var ajaxObject = new ActiveXObject( programm.version);
```

Die Objekt – Syntax für die Familie der Gecko – Browser sieht folgendermaßen aus :

```
var ajaxObject = new XMLHttpRequest();
```

Für die Microsoft – Variante ist eigens ein MSXML – Parser entwickelt worden. Im Internet – Explorer 5.0 z.B., wird die Komponente XMLHTTP verwendet.
Das Objekt wird dann wie folgt erzeugt :

```
var ajaxObject = new ActiveXObject(   Microsoft.XMLHTTP  );
```

Diese Syntax ist dann unabhängig von einer bestimmten Version des Parsers und ist somit kompatibel zu allen Versionen des Internet – Explorers. Es ist aber dennoch notwendig, bei jeder Anwendung abzufragen, mit welchem Browser der Client arbeitet. Dazu wird über eine try / catch – Anweisung nach einem Objekt gesucht und bei Erfolg eingebunden.

Es ist aber nicht nötig diese Funktion selbst zu schreiben, auch wenn sie nur aus einem geschachtelten Ausschlussverfahren besteht. Im Kapitel über die Anwendungsskripte wird dann eine von vielen Bibliotheken vorgestellt, die diese Arbeit für einen erledigt. Der Funktionsaufruf eines Ajax – Request ist also wie folgt definiert :

```
var xmlhttp = new ajaxRequest( url, (funktion) [ optionen ]);
```

Die ersten beiden Argumente (url, funktion) sind Pflichtangaben und haben folgende Bedeutung:

- url : Ist die Zieladresse für die Anfrage
- funktion: Die Funktion, die die Antwort des Servers entgegen nimmt

Der dritte Parameter ist optional und kann bestimmte Informationen enthalten, die man dem Server mitteilen möchte.

- headers : Ein Array von Wertepaaren, die im Header übergeben werden. Diese müssen der QueryString – Syntax entsprechen (Siehe Abschnitt 2.5 Parameterübergabe).
- method : die Methode, mit der der Request gesendet wird.
- body : Inhalt, der bei einem Request gesendet werden soll.
- modus : Art der Übertragung (default: asynchron).

Das komplette Zusammenspiel eines Ajax – Request hat dann folgendes schematisches Aussehen :

Beispiel.js

```
function run()
{
   new Ajax("http://localhost:8080/mache_was ,
            {onComplete: bearbeite_Response}).request();
}
```

```
function bearbeite_Response( data)
{
   var x = parseInt(data.split( "]|[")[0]);
   var y = parseInt(data.split( "]|[")[1]);
      ...   mache was damit   ...
}

if( button_gedrueckt) {
   window.setTimeout( "run()", 1000);
}
```

Erwartet man vom Server Daten, können diese im Array data abgefragt werden. Der Datenverkehr besteht aber aus reinem ASCII – Text. Damit der Browser die Daten eindeutig zuordnen kann, müssen sie sauber, durch ein Zeichen getrennt, übergeben werden. Dieses Trennzeichen wird dann mit `data.split()` lokalisiert.
Durch dieses selbst definierte Trennzeichen, kann der Browser genau feststellen, wo ein Wert (z.B. eine Zahl) aufhört und ein neuer Wert beginnt. Mit `parseInt()` wird dann der String, der zwischen zwei Trennzeichen steht, in ein Intergerwert umgewandelt. Es sollten also als Trennzeichen immer Zeichen verwendet werden, die in den erwarteten Werten möglichst nicht vorkommen können. Es gibt natürlich auch die Möglichkeit, Daten im JSON – oder XML – Format zu senden. JSON steht für **J**ava**S**cript **O**bject **N**otation und XML für **E**xtensible Markup **L**anguage. JSON – Objekte bestehen aus einer unsortierten Sammlung von Name / Value – Paaren [8], die auch geschachtelt sein können. XML – Objekte sind in der Regel grösser als gleichwertiges JSON – Objekt und ist auch aufwendiger zu erzeugen. Beide Datentransfer - Varianten stellen für mein Projekt keine Alternative dar, denn sie bieten im Hinblick auf die Datendarstellung keinen Vorteil, jedenfalls keinen Vorteil der hier nötig wäre. Eigentlich wird das Requestpaket nur größer werden.
Das Flag `onComplete` macht aus der standardmäßig eingestellten asynchronen Übertragung, eine synchrone, denn jetzt wird solange gewartet, bis die Antwort vom Server komplett übertragen wurde[13]. Eine asynchrone Übertragung ist dadurch gekennzeichnet, dass Anfragen gesendet werden können und auf die Antwort nicht gewartet wird. Wenn nun die Bearbeitung der Anfrage längere Zeit in Anspruch nimmt, kann es passieren, dass mehrere Requests abgeschickt wurden, bevor die erste Antwort die Seite aktualisiert. Die hier erstellten Anwendungen können aber mit start bzw. stop - Buttons an und – ausgeschaltet werden. Bei den Tests hat sich dann herausgestellt, dass obwohl der stop – Button betätigt wurde, die Anwendung durch noch eintreffende Daten, nachlief. Deshalb habe ich mich hier anwendungsbedingt für synchrone Datenübertragung entschieden, obwohl im Allgemeinen die asynchronen Übertragung vorzuziehen ist. `onComplete` ist allerdings keine Standard – Ajaxfunktion, sondern wird von Bibliotheken zur Verfügung gestellt (dazu später mehr). Zum Schluss wird die Funktion `window.setTimeout( "run()", 1000);` aufgerufen.
Diese ruft jetzt zyklisch jede Sekunde die Funktion run() auf; es wird also jede Sekunde eine Anfrage an den Server gestartet. Als Alternative könnte hier auch `setInterval( this.run, 1000);` verwendet werden. Die Syntax eines Ajax - Requests hat folgendes aussehen :

```
run : funktion()  {
  new Ajax("http://localhost:8080/mache_was ,
            {onComplete: bearbeite_Response}).request();
}
```

13 Eine Übertragung endet serverseitig immer mit der Zeichenkette `\r\n`

Die Realisierung von periodischen Anfragen ist somit komplett. Es muss nun noch das Funktionalität der Scripte implementiert werden.

6.4 Scripte zur Visualisierung

Als erstes sollen Grafiken auf einer HTML – Seite dynamisch angezeigt werden. Um optische Effekte zu demonstrieren und die Anwendung trotzdem einfach und überschaulich zu halten, wird als Einstieg nur mit Gif – Dateien gearbeitet.
Es existieren aber weitere Möglichkeiten um Webseiten dynamisch zu verändern z.B. CGI – Skript[14], PHP[15], Flash[16], SVG, Perl, CFML[17], ASP, JSP oder Java[18]. Ein umfassende Diskussion über sämtliche Vor – und Nachteile, Systemvoraussetzungen, Leistungskriterien bis hin zu Grenzen des Machbaren, soll hier bewußt vermieden werden. Hier wird auf die entsprechende Literatur (Fussnoten) verwiesen. Auch sind für viele der aufgeführten Techniken spezielle Serverimplementierungen nötig (z.B. der ColdFusion Application Server für CFML), die im Embedded Bereich, bzw. Echtzeit – Betriebssytemen, kein Standard sind. Dynamische Webgrafiken sind trotz dieser fehlenden Funktionalität möglich, wie das erste Beispiel zeigen wird. Später wird dann noch eine komplexere Technik vorgestellt und praktiziert (SVG). Welche Vor – und Nachteile die Art der Technik auf die Anwendung insgesamt hat, wird dann in einem gesonderten Kapitel betrachtet.

6.4.1 Dynamische Grafiken mit GIF – Dateien

Dieses Beispiel soll wie folgt ablaufen. Der Benutzer öffnet seinen Lieblingsbrowser und lädt eine Beispiel – Seite. Die Grafik kann mit zwei Buttons (stop und start) bedient werden. Startet der Anwender die Grafik wird 20 mal in der Sekunde eine Anfrage an den Server gestellt. Übergeben wird dabei ein bestimmter Winkel, der als Zähler implemen – tiert ist und beim Start bei 0 beginnt. Mit jeder Anfrage wird dieser Zähler um 1 erhöht. Der Server berechnet aus diesem Winkel die Sinus – bzw. Cosinuswerte und schickt die entsprechenden Bildschirmkoordinaten als Antwort zurück. Der Browser zeigt an dieser Stelle ein GIF – Bild (roter Punkt) an. Mit jeder Gradzahl wird wird das Bild um einen Mosaikstein erweitert. Bei 360° ist eine komplette Sinuskurve gezeichnet und der Zähler beginnt wieder bei 0. Neben den schon im letzten Abschnitt behandelten Ajax – Request, werden dafür noch nachfolgend beschriebene Funktionen benötigt :

```
var elemImg = document.createElement("img");
```

Mit dieser Methode wird ein neues XHTML – Dokument erzeugt [8]. Als Argument wird der Methode ein Name für den gewünschten Tag übergeben. Der Rückgabewert der Funktion ist ein neuer Elementknoten der in den Dokumentbaum eingehängt wird. Um das Element elemImg mit dem CSS (Cascading Style Sheets) zu verändern, kann man sich, wie oben gezeigt, den Elementbaum entlanghangeln. Einfacher geht es mit der Funktion :

```
var element = document.getElementById("img");
```

`element` ist jetzt eine Kopie von img. Würde `element` angezeigt werden, würde es das Original verdecken. Der Betrachter könnte nicht unterscheiden, welches Bild zu sehen ist.

14 http://de.wikipedia.org/wiki/Common_Gateway_Interface
15 http://de.wikipedia.org/wiki/PHP
16 http://de.wikipedia.org/wiki/Adobe_Flash
17 http://de.wikipedia.org/wiki/CFML
18 http://www.torsten-horn.de/techdocs/db-web.htm

Um die beiden Bilder unterscheidbar zu machen, werden als erstes die Attribute gesetzt.

```
element.setAttribute(   name,   wert);
```

Die Anzahl der Attribute eines Objekts ist nicht begrenzt. Es hängt im Allgemeinen davon ab, wie komplex ein Element ist oder wieviele Attribute einem einfallen und sinnvoll sind. Die hier entwickelte Sinuskurve besteht aus 360 Bildpunkten, für jede ganze Gradzahl einen. Das Attribut name würde also der Winkelwert sein und das zweite Attribut ist die Gif – Datei, die angezeigt werden soll.

```
element.setAttribute(   nummer,   bild+ gradzahl);
element.setAttribute(   image,   roter_Punkt.gif);
```

Mehr Attribute sind nicht nötig.

Das Element muss nun noch auf dem Bildschirm, an einer ganz bestimmten Position, sichtbar gemacht werden. Dazu müssen Style – Eigenschaften gesetzt werden und zwar für die Position des Objekts und seine Sichtbarkeit. Da es sich um eine Gif – Datei handelt geht es nur um sichtbar oder nicht sichtbar.

```
element.style.position = "absolute";
element.style.display = "";
element.style.top = x_wert + "px";
element.style.left = y_wert + "px";
```

Die Position ist absolut, denn sie soll sich genau dort befinden, wo es der Sinuswert festlegt. Bei Display wird der default – Wert genutzt. Wird hier nichts angegeben, wird das Bild angezeigt. Wird “none“ angegeben wird das Bild unsichtbar. Die spezielle Position wird vom Koordinatenursprung aus betrachtet. Am Ende steht das Kürzel “px“ was die Einheit in Pixel bedeutet. Damit sich eine Grafik an einer bestimmten Stelle im Browserfenster befindet, muss sie positioniert werden. Bestimmte Positionen können auf einer HTML – Seite nur mit einem Tabellenraster festgelegt werden. Ansonsten würde jede erzeugte Grafik, beim Vergrößern oder Verkleinern des Browserfensters zerstört werden. Erfragt werden kann die Höhe und Breite sowie der Koordinatenursprung links oben (top, left). Im Code sieht das dann wie folgt aus :

```
var offset_x = $("zellen_nummer").getCoordinates().left;
var offset_y = $("zellen_nummer").getCoordinates().top;
```

Die letzte Funktion dieses Abschnitts ist appendChild(). Sie wird verwendet, um einen neuen Kindknoten in die Hierarchie einzufügen. Das Element, was angehängt werden soll, wird als Argument übergeben. Im Sinus – Beispiel werden also 360 Elemente zu einer “quasi“ Liste verbunden, die dann angezeigt und wieder unsichtbar gemacht werden können.

```
dokument.body.appendChild( element);
```

An diesem Aufruf wird schon das etwas umständlich Handling der Funktionen deutlich. Im nächsten Abschnitt soll eine Bibliothek vorgestellt werden, die den Umgang mit solchen Konstrukten etwas einfacher gestalten soll.

6.4.2 Die Bibliothek “mootools.js“

Dem aufmerksamen Leser ist sicherlich das Zeichen '$' beim Erfragen der Koordinaten ins Auge gefallen. Dies passt so gar nicht in das restliche Schriftbild. Das Zeichen definiert eine Hilfsfunktion von Prototype[19] (wird auch als Bequemlichkeitsfunktion bezeichnet). Die Hilfsfunktion `$` übersetzt eine Zeichenkette in das entsprechende über `document.getElementById` adressierte Element. Wird `$` mit mehr als einem Argument aufgerufen, wird diese Substitution für alle Argumente durchgeführt und ein Array mit den Ergebnissen zurückgeliefert. Es gehört auch zum Leistungsumfang der hier eingebundenen Bibliothek MooTools [10].
JavaScript wurde 1995 mit dem Netscape – Navigator eingeführt und ist heute eine der wichtigsten, wenn nicht sogar die wichtigste, clientseitig interpretierte Scriptsprache im Web. Inzwischen wurde JavaScript als Standard anerkannt und wird in seiner Kernfunktionalität von allen gängigen Browsern unterstützt. Zudem wurde sie im laufe der Jahre um viele nützliche Eigenschaften erweitert (siehe Abschnitt HTML und DOM). Trotzdem ist die Implementierung in den Browsern bis heute nicht einheitlich.
Gerade bei sehr komplexen Anwendungen ist eine Anpassung an die unterschiedlichen Browser[20] meist nicht zu vermeiden und die damit verbundenen Tests kosten zusätzlich Zeit und Nerven. Hier setzen Bibliotheken an, indem sie immer wiederkehrende Funktionalität in einer Sammlung von Funktionen kapseln und unter einer Open Source Lizenz veröffentlichen. So wird sichergestellt, dass viele Entwickler sie benutzen und so auf vielen Plattformen testen. Inzwischen gibt es eine reichliche Anzahl von Bibliotheken. Die bekanntesten (nach meiner Recherche) sind wohl Prototype, Script.aculo.us und Yahoo UI-Library.
Eine noch relativ junge Bibliothek ist MooTools. Sie war ursprünglich eine Effektbibliothek, die auf dem bekannten Framework prototype.js aufbaute.
Nachdem sich der Funktionsumfang stetig erweiterte, wurde 2006 ein eigenes, von Prototype unabhängiges, Paket unter dem Namen mootools.js veröffentlicht. In den hier entwickelten Beispielen spürt man aber deutlich die Herkunft von Mootools. Im ersten Sinus – Beispiel wurde eine neue Anfrage mit `new Ajax` erzeugt. In JavaScript hätte entweder ein XMLHttpRequest oder ein ActiveXObject erzeugt werden müssen (siehe Abschnitt3.2.1). MooTools unterstützt also ein Klassenkonzept. JavaScript kennt aber keine Klassen. Durch das Einbinden von mootools.js werden nun sämtliche Anpassungen von der Bibliothek übernommen. Auch das Flag `onComplete` ist Bestandteil von mootools.js. Bleibt noch das Zeichen '$'. Das Dollar – Zeichen ist ein Platzhalter für die Funktion `document.getElementById( elementname)`. Zur Schreibersparnis und um nervige copy/paste – Fehler zu minimieren, kann nun `$( elementname)` geschrieben werden. Es gibt noch weitere Textersetzungen, die aber hier keine Rolle spielen. Die gesamte Bibliothek mit ausführlicher Dokumentation kann kostenfrei heruntergeladen[21] werden. Nach diesen Erläuterungen kann der Beispielcode von sinus1.js problemlos nachvollzogen werden.
Das geladene Script schickt einen Winkel zum Server und bekommt die x und y – Koordinate zurückgeliefert. Diese werden dann in die Größenverhältnisse des Koordinatensystems transformiert. Als Ausgangspunkt dient die Mitte der Grafik. Von dort wird durch geeignete Offsets, Start und – Endpunkt in Pixeln bestimmt. Durch das Flag `running` kann die Animation gestartet bzw. gestoppt werden. Als Intervall sind 50 ms eingestellt. Da sich auf der Seite mehrere Grafiken befinden, wurden die Funktionen durchnummeriert.

19 http://www.prototypejs.org/api
20 Schon bei unterschiedlichen Versionen des selben Browsers ist die Funktionalität nicht garantiert.
21 http://mootools.net/ bzw. http://docs.mootools.net/ bis jetzt nur in englischer Sprache

Beispiel: Sinus1.js

```
var winkel4 = 0;
var running4 = false;
// Client    Request
function run4() {
   new Ajax("http://localhost:8080/sinus?winkel=" + winkel4,
      {onComplete: handleKreisfktResponse4}).request();
}
// Server    Response
function handleKreisfktResponse4( data)
{
   var x = (data.split("]|[")[0]);
   var y = (data.split("]|[")[1]);

   if( !$( "img4_" + winkel4))
   {
      var elemImg = document.createElement( "img");
      elemImg.setAttribute( "id", "img4_" + winkel4);
      elemImg.setAttribute( "src", "Grafiken/Roter_Punkt.gif");
      elemImg.style.position = "absolute";
      elemImg.style.display = "";

      var offset_x = $("sinus_img_1").getCoordinates().left + 35;
      var offset_y = $("sinus_img_1").getCoordinates().top +
                     $("sinus_img_1").getCoordinates().height / 2;
      var xp = x * (430/710);
      var yp = y * 140;
      elemImg.style.left = offset_x + xp + "px";
      elemImg.style.top = offset_y - yp + "px";
      $("winkel_td_4").appendChild(elemImg);
   }
   else
   {
      var elemImg = $("img4_" + winkel4);
      /* Wenn Punkt sichtbar, dann loeschen.
         Wenn Punkt unsichtbar, dann anzeigen  */
      if (elemImg.style.display == "") {
         elemImg.style.display = "none";
      }
      else
         elemImg.style.display = "";
   }
    winkel4++;
   if( winkel4 == 720)
      winkel4 = 0;
   if( running4)
      window.setTimeout( "run4()", 50);
}
```

6.4.3 Dynamische Grafiken mit SVG

SVG steht für Scalierbare Vektor Grafiken, eine XML – Grammatik für veränderbare Grafiken, verwendbar als ein XML – Namensraum. SVG ist eine Sprache, um zweidimensionale Grafiken in XML zu beschreiben. SVG erlaubt drei verschiedene Grafikobjektarten: Vektorgrafikformen (z.B. Wege, die aus geraden Linien und Kurven bestehen), Bilder und Text. Grafische Objekte können gruppiert, gestaltet, transformiert und zusammengesetzt werden in zuvor gerenderten Objekten.

Die Palette der Eigenschaften schließt verschachtelte Transformationen, Ausschneiden von Pfaden, Alpha-Masken, Filtereffekte und Vorlageobjekte[22] ein.
SVG – Zeichnungen können interaktiv und dynamisch sein. Animationen können entweder deklarativ (z.B. durch die Einbindung eines SVG – Animationselements in den SVG – Inhalt) definiert und ausgelöst werden oder über Skripte.
Hoch entwickelte SVG-Anwendungen sind mit Hilfe einer ergänzenden Skript – Sprache möglich, die Zugriff auf das SVG Document Object Model (DOM)[23] von SVG hat und vollen Zugang zu allen Elementen, Attributen und Eigenschaften. Eine Vielzahl der Event-Handler wie `onmouseover` oder `onclick`, können jedem SVG – Grafikobjekt zugeordnet werden. Auf Grund seiner Kompatibilität und dem Einfluss auf andere Web – Standards, können Dinge wie Skripting gleichzeitig auf XHTML – und SVG – Elemente innerhalb der gleichen Webseite angewendet werden.
Ein SVG – Dokument beginnt immer mit einer Dokumenttyp – Deklaration. In ihr müssen bestimmte Identifizierer eingetragen werden, die ein SVG – Dokument kennzeichnen.

- SVG – Namensraum: http://www.w3.org/2000/svg
- Identifizierer SVG 1.0: PUBLIC "-//W3C//DTD SVG 1.0//EN"
- System – Identifizierer:
  http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd

Alles in allem hat ein weißes Blatt in SVG folgende Struktur:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="20cm" height="30cm" xmlns="http://www.w3.org/2000/svg">
  <desc>Ein leeres Din A4 Blatt
  </desc>
   ...   Hier koennen jetzt alle Grafiken gezeichnet und animiert
              werden   ...
</svg>
```

Das weitere Vorgehen hat ab jetzt große Ähnlichkeit mit der Syntax des letzten Abschnitts (HTML, Ajax und JavaScript). Deshalb werde ich mich hier auf die grundlegende Funktionalität der hier entwickelten Anwendungen beschränken. Ein hervorragendes Tutorial in deutscher Sprache, was den gesamten Sprachumfang von SVG abdeckt und das auch hier die Grundlage bildet, ist unter http://svg.tutorial.aptico.de/ zu finden.

6.4.4 Einbetten des Ajax – Request

Ob SVG überhaupt eine Alternative für die Entwicklung von dynamischen Webseiten darstellt, hängt in erster Linie davon ab, ob automatische Anfragen an den Server unterstützt werden. SVG bietet die Möglichkeit den entwickelten Ajax – Request einzubetten.

22 http://www.fh-wedel.de/~si/praktika/MultimediaProjekte/SVG/SVG_Tutorial_mi3794/2_6_2.htm
23 Was das DOM ist und wie der Zugriff realisiert wird ist in Abschnitt 6.2 beschrieben

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="20cm" height="30cm" xmlns="http://www.w3.org/2000/svg">
  <desc>Einbetten des Ajax    Request
  </desc>

  <script type="text/javascript" xlink:href="mootools.js" />
  <script type="text/javascript" xlink:href="ajax_anfrage.js" />

   ...   Hier koennen jetzt alle Grafiken gezeichnet und animiert
werden   ...
</svg>
```

Mootools spielt hier aber nur noch eine untergeordnete Rolle und ist rein auf den kurzen Ajax – Request beschränkt. Der Zugriff auf SVG–Elemente und deren Eigenschaften kann mit Mootools nicht realisiert werden. Dieser DOM – Zugriff hat folgende Syntax:

```
var but = document.getElementById('button'); oder
var but = svgDocument.getElementById('button');
```

Beide Schreibweisen sind identisch und führen nicht zu einem Syntaxfehler. svg als Präfix wurde früher nur verwendet, um dem Browser Informationen zur Darstellung von Grafiken mitzugeben. In den aktuellen Browser – Versionen ist diese Unterscheidung nicht mehr nötig. Die häufigsten Elementzugriffe, die in den hier entwickelten Beispielen zum Einsatz kommen, sind im Anhang beschrieben.
Um mit der SVG – Seite auch interaktiv arbeiten zu können, werden noch einige Ereignisse benötigt, auf die das Programm reagieren soll. Um auf die Objekte des DOM zugreifen zu können, muss immer zuerst ein benanntes Objekt für das SVG Dokument (das document-Objekt), in Form einer Variablen, erzeugt werden:

```
var document = evt.getTarget().getOwnerDocument();
```

Das Objekt `evt` repräsentiert das eingetretene Ereignis. Die Methode `getTarget()` greift auf den Knoten zu, der dieses Ereignis ausgelöst hat und liefert ihn zurück. Die Methode `getOwnerDocument()` liefert die Wurzel des auslösenden Knotens zurück, also das Objekt, welches das Dokument repräsentiert. Bei den hier erstellten Anwendungen werden die Events von der Mouse ausgelöst[24].
Zum Abschluss dieses Abschnitts soll ein kleines Beispiel [11] aus der erstellten Beispielanwendung die letzten Unklarheiten beseitigen. Es geht darum, einen Schieberegler[25] (slide) zu erstellen, mit dem eine RGB –Farbe in ihrer Intensität verändert werden kann. Die gesamte Anwendung besitzt 3 dieser Regler, je einen für Rot, Grün und Blau. Aber hier sollte einer genügen, denn letztlich sind sie (bis auf den Farbnamen) alle identisch. Die Regler werden mit der Mouse bedient (siehe Events) und liefern über die Y – Koordinate (senkrechte Bewegung) eine Verschiebung von 0 bis 100 Pixeln (entspricht der Farbintensität von 0 bis 100 Prozent). Aus der Stellung aller drei Regler wird dann die RGB – Mischfarbe berechnet und in einem separaten Rechteck angezeigt. Diese Farbe könnte so als Sollwert für eine Farbmischanlage gelten. Zuerst die SVG – Grafik eines Reglers mit einigen kleinen Erläuterungen.

24 Auflistung der Mouse – Events im Anhang
25 Beispiel aus J. David Eisenberg SVG Essentials, O'Railly, 2002

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="20cm" height="30cm" xmlns="http://www.w3.org/2000/svg">
  <desc>Dynamik mit SVG</desc>

  <script type="text/javascript" xlink:href="ajax_anfrage.js" />

   <g onmousedown="startColorDrag(0)"
      onmouseup="endColorDrag(0)"
      onmousemove="doColorDrag(evt, 0)"
      transform="translate( 900, 10)">
      <rect x="5" y="0" width="10" height="100" style="fill: red;"/>
      <line id="slide0" x1="-5" y1="0" x2="25" y2="0"
            style="stroke: gray; stroke-width: 5;" />
    </g>
</svg>
```

Neu ist eigentlich nur der Teil, der innerhalb des <g> - Tags steht. Dieses g steht für Grafik – Objekt. Alles was in diesem Symbol eingeschlossen ist wird als Einheit behandelt. Alle Koordinaten sind dann relativ zur Objektposition zu sehen. Die Position auf dem Bildschirm wird mit dem **`transform`** – Attribut festgelegt. Alle anderen x/y – Positionen beziehen sich auf dieses Bildschirm Areal. Daher sind auch negative Werte gültig. Neben der Grafik wird noch auf Mouse – Events reagiert. Es sind die Events Mouse links gedrückt, Mouse links losgelassen und Mausbewegungen über dem Objekt.
Damit wird die y – Position des Regelbereich bestimmt. Die Parameter, die den Skript – funktionen mitgegeben werden, beziehen sich auf die Nummer des gerade bewegten Reglers. Und so sehen die zugehörigen Skript – Funktionen aus.

```
var slideChoice = -1;
var colorstr;
var rgb = new Array( 100, 100, 100);
/**
  *   Legende:    0  -->      Rot
  *               1  -->      Gruen
  *               2  -->      Blau
*/
function startColorDrag( welcher)
{
      endColorDrag();
      slideChoice = welcher;
}
/**
  * Wenn der Regler verlassen wird, muss die Mouse-Auswahl
  * auf einen ungueltigen Wert (-1) gesetzt werden. 0 ist ein
  * korrekter Wert
*/
function endColorDrag()
{
      slideChoise = -1;
}
```

```
/**
  *   Zuweisen des mit der Mouse ausgewaehlten Reglers
*/
function doColorDrag( evt, welcher)
{
   if( slideChoice < 0 || slideChoice != welcher)
      return;

   /* Die relative Mouse-Position zum Reglerbalken von oben*/
   var obj = evt.getTarget();
   var pos = evt.getClientY() - 10;
   /* Regelbereich pruefen    */
   if( pos < 0)
      pos = 0;
   if( pos > 100)
      pos = 100;

   /* Schieben des Reglers an die neue Mouse-Position*/
   obj = svgDocument.getElementById( "slide" + slideChoice);
   obj.setAttribute( "y1", pos);
   obj.setAttribute( "y2", pos);

   /* neuen Farbwert berechnen      */
   rgb[slideChoice] = 100 - pos;
   /* Neuen Farbwert aus den prozentualen Reglerwerten berechnen
      Spaeter sollen die drei Werte dem server uebergeben werden:
      return farbe;     colorStr = farbe  */
   colorStr = "rgb(" + rgb[0] + "%," + rgb[1] + "%," + rgb[2]+ "%)";
   obj = svgDocument.getElementById( "Muster");
   obj.setAttribute( "fill", colorStr);
}
```

Der gesamte Code mit den restlichen Reglern und den anderen Anwendungen auf der SVG – Seite ist wieder auf der beiliegenden CD zu finden. RGB ist das invertierte Farbsystem. Das Farbrechteck Namens `Muster` ist weiß, wenn alle drei Farben volle Intensität besitzen (100 % = RGB – Wert 255). Soll jetzt z.B. Rot volle Intensität bekommen, müssen Grün und Blau nach Null verschoben werden.

7 Server auf verschiedenen Plattformen

Das Verhalten des Servers soll nun auf verschiedenen Plattformen getestet werden. Die erstellte Funktionalität wird sich auf allen Systemen identisch verhalten. Die Client / Server – Kommunikation allerdings läuft unter der Kontrolle des Betriebssystems und zwar auf beiden Seiten. Daher sind auf den unterschiedlichen Systemen einige kleine bis recht große Anpassungen vorzunehmen. Das Original wurde unter Linux erstellt und wird folgendermaßen mit dem GNU C++ – Compiler kompiliert und gestartet.

7.1 Installation auf Debian – Linux

Der gesamte Projekt – Ordner wird als erstes ins Home – Verzeichnis kopiert. Zum komfortablen Kompilieren wird dann ein Script erstellt mit Namen **server_compile**, ohne Extension, mit folgenden Inhalt :

```
#!/bin/bash
g++ -o WebServer  ../01_Main/01_main.cpp \
                  ../02_Prozesse/02_web_prozess.cpp \

                  ...   Alle weiteren Dateien   ...

                  ../07_Anwendungen/02_ajaxUhr.cpp -lpthread
```

```
Rechtevergabe, kompilieren und ausführen

home@linux:~> cd chmod +x server_compile
home@linux:~> ./server_compile
home@linux:~> ./WebServer

Socket erfolgreich angelegt
```

Alle HTML – Seiten, die aufgerufen werden sollen, müssen im selben Verzeichnis liegen wie die ausführbare Datei des Servers. Eine schon vorhandene Testanwendung heißt ajaxUhr.html. Der Server wartet auf Port 8080 auf Anfragen. Jetzt kann der Lieblings – Browser geöffnet werden und folgende URL eingegeben werden.

```
http://localhost:8080/ajaxUhr.html
```

Wenn alles korrekt verlaufen ist, sollte die aktuelle Systemzeit zu sehen sein.

7.2 Installation auf QNX

Die Inbetriebnahme auf der QNX – Plattform ist ähnlich unspektakulär. In eine Eclipse – Entwicklungsumgebung wurde das QNX – Momentics eingebettet und mit einem GNU Compiler verknüpft. In den Voreinstellungen wird Name, IP – Adresse und Portnummer angegeben. Danach kann die Applikation lokal oder remote gestartet werden. In der Rubrik “Extra Libraries“ müssen die systemeigenen Bibliotheken ph, phrender, m (für math.h) und socket eingebunden werden.
Nach Abschluss der Serverentwicklung kann die ausführbare Datei (Datei mit Extension _g) per FTP – Server dauerhaft auf das Zielsystem geschoben werden. Der Server kann jetzt direkt auf dem QNX – Rechner gestartet werden.

7.3 Installation auf Linux – RTAI

Am Anfang steht natürlich die Installation des RTAI – Echtzeitkernels auf einer bestehenden Linux – Distribution. Auf meinem Rechner ist Debian 4.1 installiert. Die Version des Linux - Kernels ist die Version 2.6.19.
Demzufolge muss ein Echtzeitkernel – Patch kompatibel zu Linux 2.6.19 ausgewählt werden. Es ist also ein ganz normaler Kernel, der nur gepatcht, d.h. modifiziert wird. Auf www.rtai.org ist die neueste Version von RTAI, die Version 3.5 (Der Download der rtai Datei 3.5 enthält dann den Patch für den Linux Kernel 2.6.19). Die Installation ist je nach Version verschieden und muss dementsprechend angepasst werden (Es muss der richtige Patch für den installierten Linux kernel ausgewählt werden). In meinem Fall wurde wie folgt vorgegangen. Zuerst wird das entsprechende Paket (Der Linux Kernel von www.kernel.org) heruntergeladen und entpackt.

```
# cd /usr/src
# tar xvjf linux-2.6.19.tar.bz2
# ln -s linux-2.6.19 linux
```

Jetzt kann der Standard Kernel (ohne RTAI) erstmal konfiguriert und getestet werden.
Danach kann RTAI entpackt werden

```
# tar -zxf rtai-3.5.tar.gz
# ln -s rtai-3.1.1 rtai
```

Dann den Standard – Kernel mit dem entsprechenden Patch aus dem RTAI Archiv modifiziernen.

```
# cd /usr/src/linux
# patch -p1 < ../rtai/rtai-core/arch/i386/patches/hal6c1-2.6.19.patch
```

Jetzt können die RTAI – spezifischen Sachen konfiguriert werden. Wem das zu kompliziert oer zu umständlich ist, übernimmt die Standardeinstellungen

```
# make menuconfig
```

Ist die Auswahl beendet kann der Kernel kompiliert werden.

```
# make
# make modules_install install
```

Ist bis hier hin alles korrekt verlaufen, können die RTAI User Space Komponenten kompiliert und installiert werden. Es ist im Prinzip dieselbe Prozedur zu wiederholen.

```
# cd /usr/src/rtai
# make menuconfig
```

Im Menü können jetzt wieder persönliche Einstellungen vorgenommen werden oder man übernimmt die Default – Belegung. Anschließend kann kompiliert und installiert werden.

```
# make            // kompilieren
# make install    // installieren
```

Ist alles ohne Fehler durchgelaufen muss der Rechner heruntergefahren und neu gestartet werden. Jetzt stoppt der Rechner beim Hochfahren und man kann das gewünschte Betriebssystem auswählen und danach den Boot – Vorgang fortsetzen.
Nun kann ein erster Test erfolgen. RTAI stellt dazu spezielle Files im Ordner latency bereit.

```
# cd /usr/realtime/testsuite/kern/latency/
# ./run
```

Auf der Home – Seite von Debian (RTAI) können noch weitere Beispiele heruntergeladen werden, um die Installation gründlich zu prüfen. Sollten diese Tests allerdings zeigen, das die Installation doch nicht ganz optimal verlaufen ist, gibt es neben dem Installationsmanual noch eine Rubrik “Troubleshooting“, in der die Behebung der gängigsten Fehler beschrieben sind. Dennoch sollte erwähnt werden, dass schon einige Erfahrung in Sachen Linux nötig sind, um flüssig durch die Installation zu kommen.

7.3.1 RTAI – Programmierung

Linux ist, wie die meisten anderen Betriebssysteme auch, nicht für Echtzeitbedingungen ausgelegt. Das bedeutet, das nicht garantiert werden kann, dass ein Programm innerhalb eines definierten Zeitfensters, Prozessorzeit zugewiesen bekommt. Im normalen User – Betrieb ist dieses Verhalten völlig ausreichend[26]. Es gibt aber Bereiche und Anwendungen, wo garantiert werden muss, dass ein System innerhalb einer bestimmten Frist reagiert. Diese Möglichkeit bietet RTAI (**R**eal **T**ime **A**pplikation **I**nterface). Dazu bietet RTAI drei Arten von Prozesse an :

1. Periodische Tasks: Das sind Prozesse, die in regelmäßigen Abständen zur Ausführung gebracht werden. Es sollte darauf geachtet werden, dass der Prozess vor dem nächsten Aufruf seine Arbeit abgeschlossen hat.
2. Sporadische Tasks: Diese Tasks werden durch ausgelöste Ereignisse gestartet, in der Regel durch Interrupts. Hier werden harte Reaktionszeiten verlangt.
3. Aperiodische Tasks: Asynchrone Intervalle für “weiche“ Bedingungen.

Welche Tasks für die Anwendungen des Webservers die günstigsten Leistungsmerkmale versprechen wird sich im weiteren Verlauf herauskristallisieren.
RTAI an sich ist kein eigenständiges Betriebssystem, sondern lässt Linux in einem eigenen Prozess laufen. Linux ist dabei der Idle – Task, der nur dann zum Einsatz kommt, wenn kein anderer Echtzeitprozess lauffähig ist. Man könnte auch sagen, der Linux – Prozess hat die geringste Priorität. Alle Interrupts werden von RTAI abgefangen und erst wenn keiner für RTAI bestimmt ist, werden sie an Linux weitergereicht.
Da RTAI – Programme als Kernelmodule arbeiten, sind sie völlig anders aufgebaut als “normale“ C – Programme. Sie besitzen keine main – Funktion und sämtliche Standard Bibliotheken können nicht verwendet werden. Das bedeutet u.a., dass mit `printf()` keine Ausgaben auf dem Bedienterminal möglich sind. Sämtliche Beispiel – Examples behaupten zwar mit der Verwendung von `printk()` (für print Kernel) oder `rt_printk()` (für Real – Time print) das Problem behoben zu haben, doch ich konnte auf meinem Terminal keine Ausgaben erkennen. Nach intensiven Recherchen[27] wurde klar, dass Kernel – Module keine Ausgabe auf dem Bedienterminal machen. Denn Kernel – Module schreiben in die Kernellog – Datei. Wenn das nicht so wäre, könnte man auf der Befehls – ebene überhaupt nicht effektiv arbeiten, da ständig Kernel – Informationen dazwischen funken würden.
Ohne Standard – Bibliotheken ist natürlich die Funktionalität von RTAI – Programmen stark eingeschränkt[28] und sollten sich daher nur auf den zeitkritischen Teil beschränken.

26 Es sei denn man möchte bei EBay garantiert das letzte Gebot abgeben.
27 http://www.captain.at/rtai-manual.php
28 sämtliche Systemfunktionen wie socket(), accept(), send(), recv() usw. stehen nicht zur Verfügung

Dies ist möglich, da RTAI der darunterliegenden Schicht (Linux) erlaubt, alle seine Funktionen und Dienste beizubehalten. Folglich muss, wegen der fehlenden Funktionalität, der gesamte Server auf der Linux – Seite verbleiben, während die erstellten Beispiel – Anwendungen auf die RTAI – Seite wandern. Zwischen beiden Teilen muss über entsprechende Kommunikationskanäle eine Brücke geschlagen werden, damit wie bisher, der Datenfluss aufrecht erhalten werden kann. Dazu werden Funktionen zum Erzeugen von Prozessen, Funktionen zur Steuerung von zeitlichen Abläufen und Funktionen zur Kommunikation benötigt, die im nachfolgenden Abschnitt ausführlich beschrieben werden. Für alle anderen Funktionen sei auf die RTAI – Seite http://www.rtai.org/ verwiesen.

7.3.2 RTAI – API

Zu Beginn soll der grundsätzliche Aufbau eines RTAI – Moduls beschrieben werden. Gegeben sei also folgendes Code – Fragment :

```
/*modul1.c*/
# define MODULE
# include <linux/module.h>

int init_module(void)
{
   /*   ...   Modul-Funktionalitaet   ...   */
   return 0;
}
void cleanup_module( void)
{
   /*   ...   Aufraeumarbeiten   ...   */
   return;
}
```

1. Die Präprozessoranweisung define MODULE sorgt dafür, dass der für die Modul- und Kernelprogrammierung notwendige Anteil in den Headerdateien Verwendung findet.
2. Die includierte Headerdatei <linux/module.h> ist die Standard – Bibliothek für das Erzeugen von Modulen
3. *init_module:* wird beim Starten aufgerufen (insmod) und initialisiert das Modul/ den Treiber...
4. *cleanup_module:* wird beim Entfernen des Moduls ausgeführt (rmmod). Sie informiert den Kern, damit er den Speicher bereinigt und alle allokierten Ressourcen wieder freigibt

Besteht das Modul aus mehreren Dateien, wird zunächst jede Datei für sich kompiliert, was eine Objektdatei “Modul.o“ ergibt und dann werden alle zu einer ausführbaren Datei zusammengefügt. Für den hier zugrunde liegenden Kernel 2.6 ist ein spezieller Makefile vorgeschrieben.

```
obj-m := Modul.o
KDIR  := /lib/modules/$(shell uname -r)/build
PWD   := $(shell pwd)
default:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

Zum Schluss wird noch ein Script benötigt, das runinfo heißt und meist als versteckte Datei ausgewiesen ist. Sie beinhaltet wichtige Informationen für die rtai-load – Datei, die sich im Verzeichnis `/usr/realtime/bin` befindet und bildet letztlich die Schnittstelle zwischen RTAI – Programm und Benutzer – Programm. Das User – Programm ist dann der erstellte Webserver und wird später zwischen den beiden Semikolons eingetragen.

```
latency:ksched+fifos:push Modul;;popall:control_c
```

Alle Funktionen der RTAI – API die hier zum Einsatz kommen sind im Anhang aufgeführt und mit einer kurzen Erläuterung versehen.
Um jetzt ein genau definiertes und gewünschtes Zeitverhalten der einzelnen Tasks zu erreichen, stellt RTAI Timer zur Verfügung. Auch diese werden hier nur benutzt. Für die genauen Erklärungen sei wiederum auf den Anhang verwiesen. Ein kurzes Beispiel sollte genügen um sich einen grundsätzlichen Überblick zu verschaffen.Damit es kein sinnloses Beispiel wird, soll die Zeit gemessen werden, die ein Prozess für die Erledigung seiner Aufgaben benötigt. Dies ist wichtig zu wissen, damit die Ausführungsperiode eines Prozesses nicht zu kurz gewählt wird. Später im Kapitel wird dieses Beispiel zur Messung der Leistungsdaten wieder verwendet werden.

```
RT_TASK task;
void prozess()
{
   int i, j, k;         // Durch for()-Schleifen Zeit verbrauchen
   for( i=0; i<1000; i++)
   {
      for( j=i; j<1000; j++)
         k = i*j;
   }
}
int init_module( void)
{
   RTIME start, stop, zeit, zeit_nano;
   /*   Prozess erzeugen   */
   rt_task_init( &task, prozess, 0, 4000, 0, 0, 0);
   /*   Aktuelle Zeit lesen   */
   start = rt_get_time();
   /*   Den erzeugten Prozess starten   */
   rt_task_resume( &task);
   /*   Nach der Rückkehr des Prozess wieder Zeit lesen   */
   stop = rt_get_time();
   /*   Zeitdifferenz berechnen   */
   zeit = stop - start;
   /*   konvertieren von Timer    Ticks in Nanosekunden   */
   zeit_nano = count2nano( zeit);
   /*   Ergebnis steht in der Kernellod - Datei   */
   rt_printk(   \n\tDie Ausführungszeit beträgt %d ns\n, zeit_nano);
   return 0;
}
void cleanup_module( void)
{
   /*   Aufräumarbeiten   */
   rt_stop_timer();
   rt_task_delete( &task);
   return;
}
```

7.3.3 RTAI – Kommunikation (FIFO's)

Der Begriff FIFO steht für First in First out und sagt aus, dass Daten seriell nach dem Warteschlangen – Prinzip geschrieben und gelesen werden. Eine RT – FIFO ist ein Prozess, der die Kommunikation zwischen Linux und Realzeitsystem RTAI erlaubt. Die Daten die zuerst abgelegt werden, werden auch als erstes wieder ausgelesen. Im Prinzip ist ein FIFO eine spezielle Datei unter Linux, die zwar im Dateisystem existiert, sich jedoch so verhält, als wäre sie eine benannte Pipe. Sie stellt also nur einen globalen Datenpuffer zwischen zwei Programmen dar. Über diesen Puffer können dann Daten ausgetauscht werden. Neben den “normalen“ FIFOs existieren noch die so genannten Realzeit – Fifos, die hier zum Einsatz kommen sollen. Diese FIFOs erlauben es nicht, das ein Echtzeit – Prozess blockiert wird, der in eine volle FIFO hineinschreiben will oder aus einer leeren lesen möchte. Diese FIFOs werden auch nicht aus Platzmangel auf die Festplatte ausgelagert. Normale FIFOs im User – Programm werden folgendermaßen behandelt. Zum besseren Verständnis hier noch mal grafisch das Prinzip eines FIFOs. Dieser Grafik liegt schon das spätere zu realisierende Kommunikationsprinzip zwischen Server und RTAI zu Grunde.

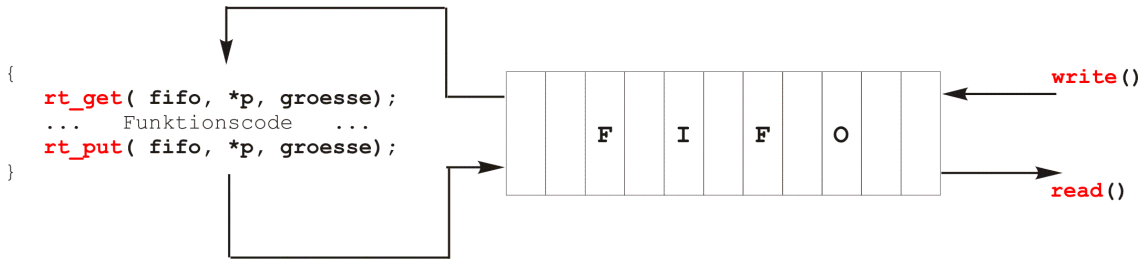


Abbildung 8: Kommunikationsprinzip von Echtzeit - FIFOs

Im nächsten Schritt werden die Funktionen des FIFOs auf der RTAI – Seite behandelt. Ihre Handhabung ist etwas umfangreicher. Wie bereits weiter oben erwähnt ist ein Realtime – FIFO in einem Realtime – Prozess eingebettet, der periodisch, aperiodisch oder spontan ausgeführt werden kann. Ich werde mich hier für die spontane Variante entscheiden, denn der Prozess soll nur angestoßen werden, wenn Daten angefordert werden. In der Realität würde der Prozess periodisch laufen und ständig Werte abfragen. Wenn dann eine Anfrage von Server kommen würde, würden die aktuellen Daten in den FIFO geschrieben. Für den sporadischen Fall wird ein FIFO – Handler benötigt, der den FIFO ständig beobachtet. Wie ein FIFO – Handler eingerichtet wird und welche Funktionen nötig sind steht wieder im Anhang.

7.3.4 Server / RTAI – Kommunikation

Das Konzept, was es jetzt zu realisieren gilt, ist in Abbildung 9 grafisch dargestellt.

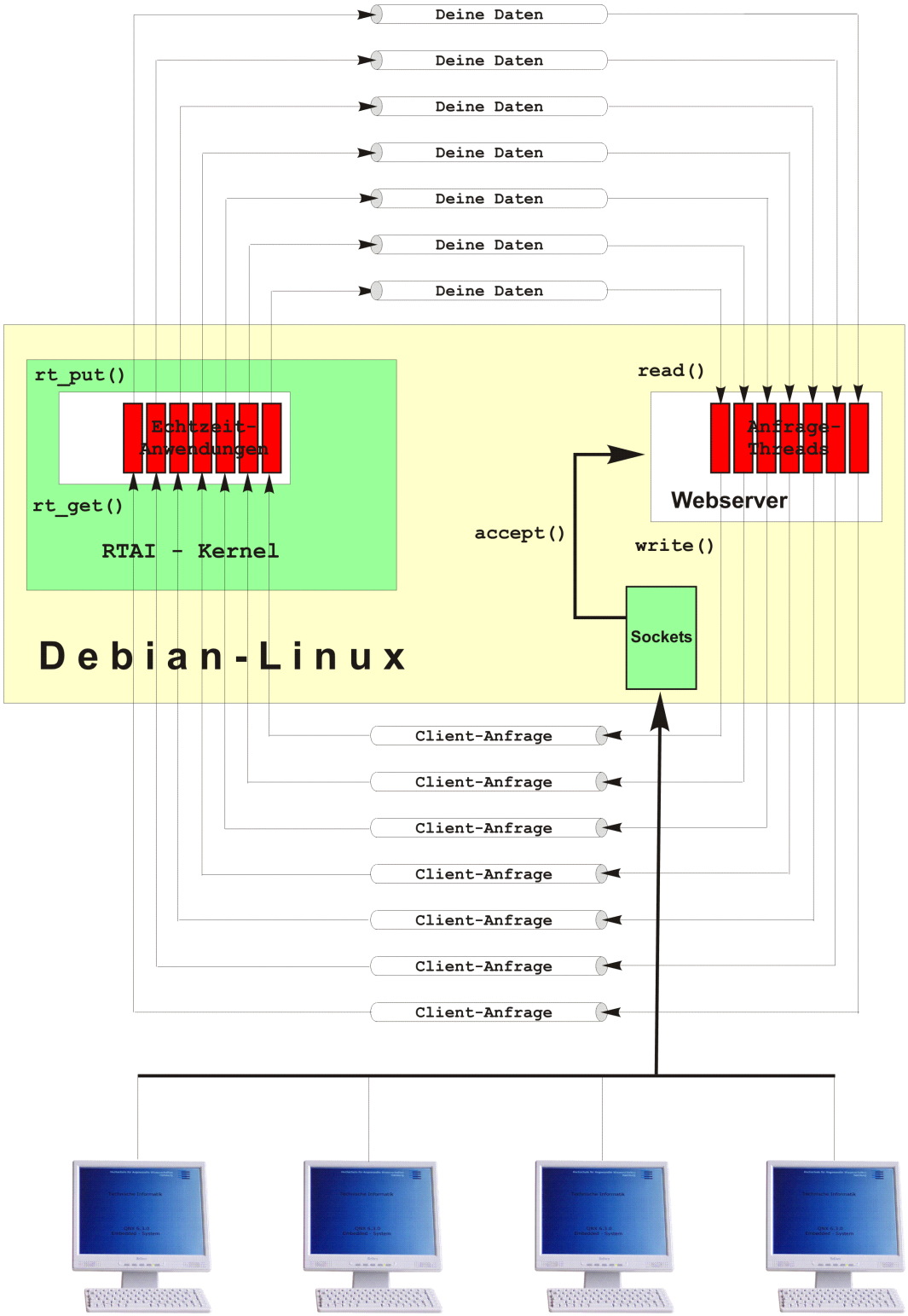


Abbildung 9: Zu realisierende Aufgabenstellung

Die Kommunikation zwischen RTAI und Linux kann hier nicht mit herkömmlichen FIFOs realisiert werden. Da die unter RTAI erstellten Anwendungen ein Kernel – Modul darstellt, wird für den Datenaustausch ein Gerätetreiber benötigt.
Die Kommunikation wird also so behandelt, als handle es sich um die Kommunikation zwischen Betriebssystem und Hardware. Unter Linux ist dieser Treiber immer ein Teil des Kernels und kann entweder statisch hinzugelinkt oder auf Anfrage des Kernel – Moduls nachgeladen werden.

```
/root> rtaiDev.sh
```

Da ein Gerätetreiber aber ein Teil des Kernels ist, kann aus Sicherheitsgründen nicht aus einem normalen User – Prozess (unser Server) darauf zugegriffen werden. Um dennoch auf Hardware – Komponenten (Festplatte, CD – ROM, Soundkarte oder eben RT – FIFOs) zugreifen zu können, wurde unter Linux / Unix ein Mechanismus eingeführt, mit dem recht einfach, über eine "Hardware – Gerätedatei", mit einem Gerätetreiber kommuniziert werden kann. Diese "Datei" kann mit einem gewöhnlichen Prozess geöffnet werden und lesend bzw. schreibend auf sie zugegriffen werden. Man kann also mit den normalen E/A – Operationen (open(), read(), write(), close()) über diese "Datei" auf einen Gerätetreiber zugreifen. Diese Operationen sind im Anhang beschrieben. Der Anwendungsteil auf der User – Seite beschränkt sich auf das Weiterreichen der Übergabeparameter vom Client und ist daher recht übersichtlich. An einer von mehreren erstellten Anwendungen, soll exemplarisch das Zusammenspiel demonstriert werden. Ich werde mich dazu wieder der Systemzeit bedienen.

```
class SVG_Uhr : public Anwendung
	{
	private:
		HTTP_Header header;
		ListenEintrag *le;
		time_t zeit;
		char daten[255];
		struct client_to_rtai
		{
		   int radius, mittelpunkt_x, mittelpunkt_y;
		   int stunde, minute, sekunde;
		}
		struct rtai_to_client
		{
		   int stunde_x, stunde_y;
		   int minute_x, minute_y;
		   int sekunde_x, sekunde_y;
		}
		client_to_rtai *cr;
		rtai_to_client *rc;
		char temp[5];
	protected:
	public:
		SVG_Uhr();
		virtual ~SVG_Uhr();
		void oj_send_daten( int socket, QueryString *qs);
	};
```

Wie die Klassendefinition schon erahnen lässt, schickt der Client den Mittelpunkt der Uhr und den Radius des Ziffernblattes. Die Serveranwendung packt die aktuelle Systemzeit in Zahlen hinzu und übergibt alle Daten dem Echtzeit – Modul.

Auf die Darstellung von Konstruktor und Destruktor soll an dieser Stelle verzichtet werden. Sie beschränken sich auf die Allokierung von Speicher für die Datenpakete, Initialisierungen und das Erfragen der Systemzeit.
Es ist allerdings zu beachten, dass hier mit einem Echtzeit – Modul hoher Priorität kommuniziert wird und das ein FIFO – Handler aktiviert ist, der bei jeder Veränderung auf der Datei, die Echtzeit – Anwendung aufruft[29]. Die Anwendung auf der User – Seite muss also je eine Gerätedatei zum Schreiben und Lesen öffnen. Auch eine FIFO erst zum Schreiben zu öffnen, danach zu schreiben und anschließend zum Lesen zu öffnen, um die Ergebnisse zu erhalten, ist kein Ausweg. Denn die User – Seite kann die Echtzeit – Anwendung nicht unterbrechen um den FIFO – Modus zu wechseln.
Hier als erstes die Anwendung auf der User – Seite (Server) :

```
void SVG_Uhr::oj_send_daten( int socket, QueryString *qs)
{
   int i;
   ctime( &zeit);       // Systemzeit holen
   for( le = qs->anfang, i = 0; le; le = le->next, i++)
   {
      if( i == 0)
         cr->radius = atoi( le->get_wert());
      else if( i == 1)
         cr->mittelpunkt_x = atoi( le->get_wert());
      else if( i == 2)
         cr->mittelpunkt_y = atoi( le->get_wert());
   }

   /* Umwandeln des Zeitstrings in Integer    Werte
      fuer Stunde, Minute und Sekunde */

   /*   Echtzeit-Fifos einrichten   */
   int senden = open(  /dev/rtf0, WRONLY);
   int empfangen = open(  /dev/rtf1  ,RDONLY);

   /*   Daten in die Fifo schreiben   */
   read( senden, cr, sizeof( client_rtai));
   /*   Antwort lesen   */
   write( empfangen, rc, sizeof( rtai_client);

   /*   Fifos schliessen   */
   close( senden);
   close( empfangen);

   /*   HTTP    Paket packen und zurueck zum Clienten   */
   sprintf( daten, "%s]|[%d]|[%d]|[%d]|[%d]|[%d]|[%d", &zeit,
            rc->stunde_x, rc->stunde_y, rc->minute_x, rc->minute_y,
            rc->sekunde_x, rc->sekunde_y);
   header.send_HTTP_Header( socket, 200, "SVG_Uhrzeit",
                            strlen( daten) + 1, NULL);
   header.sendblock( socket, daten, strlen( daten));
   header.sendpuffer( socket, "\r\n\r\n");
   free( cr);
   free( rc);
}
```

29 Die Echtzeit – Anwendung wird also zweimal aufgerufen, bei rtf_get() und rtf_put(). Wird eine Fifo im R/W – Modus betrieben, werden auf der User – Seite die eigenen Daten gelesen, die gerade geschrieben worden sind.

Die RTAI – Seite, welche die Daten vom Server entgegennimmt, ist folgendermaßen aufgebaut. Auf das Init – und Cleanup – Modul wird an dieser Stelle verzichtet. Für den kompletten Quellcode sei wieder auf die beiliegende CD verwiesen. Ich beschränke mich hier auf die Funktionalität des FIFO – Handlers und die Echtzeitanwendung.

```
# include <linux/kernel.h>
# include <linux/module.h>
# include <asm/io.h>
# include <math.h>
# include <rtai.h>
# include <rtai_sched.h>
# include <rtai_fifos.h>

# define EMPFANGSPUFFER struct eingangswerte_svg_uhrzeit
EMPFANGSPUFFER
{
      int radius;
      int mittelpunkt_x, mittelpunkt_y;
      int stunde, minute, sekunde;
};

# define SENDEPUFFER struct returnwerte_svg_uhrzeit
SENDEPUFFER
{
      int mittelpunkt_x, mittelpunkt_y;
      int stunde_x, stunde_y;
      int minute_x, minute_y;
      int sekunde_x, sekunde_y;
};
```

Die Funktion `Msg_Handler( int t)` wird vom Echtzeit – Modul jede msec gestartet. Er horcht auf dem Fifo rtf5, ob Daten zum Lesen auf dem Fifo geschrieben worden sind.

```
/*****************************************************/
/*   ---   ---      Message - Handler     ---   ---   */
/*****************************************************/
void Msg_Handler( int t)
{
      rtf_create( 5, 100);    // Lese-Fifo /dev/rtf5 initialisieren
      rtf_create( 6, 100);    // Schreib-Fifo /dev/rtf6 initialisieren
      rtf_create_handler( 5, SVG_Uhrzeit);
}
```

Sind Daten vom Server in den Fifo geschrieben worden, wird SVG_Uhrzeit gestartet.

```
/*   Systemzeit   */
SENDEPUFFER       sp;
EMPFANGSPUFFER    ep;

int SVG_Uhrzeit()
{
      int zw_schritt;
      double sinus, cosinus, bogenmass;
      sinus = cosinus = bogenmass = 0;

      rtf_get( 5, &ep, sizeof( EMPFANGSPUFFER)); // Daten lesen
```

```
        // Die Mittelpunkt - Koordinaten bleiben unverändert
        sp.mittelpunkt_x = ep.mittelpunkt_x;
        sp.mittelpunkt_y = ep.mittelpunkt_y;

        // Berechnung der Koordinaten des Stundenzeigers
        zw_schritt  = ep.minute / 15;            // Viertelstunden
        bogenmass   = ( (((ep.stunde % 12) * 30) + (zw_schritt * 8)) *
                          3.141592654) / 180;
        sinus       = rt_sinus( bogenmass);
        cosinus     = rt_cosinus( bogenmass);
        sinus       = (ep.radius - 40) * sinus;
        cosinus     = (ep.radius - 40) * cosinus;
        sp.stunde_x = ep.mittelpunkt_x + (int)sinus;
        sp.stunde_y = ep.mittelpunkt_y - (int)cosinus;

        rtf_put( 6, &sp, sizeof( SENDEPUFFER));  // Antwort schreiben
        return 0;
}
```

Abgebildet ist hier aus Platzgründen nur der Code des Stundenzeigers. Der Code für den Minuten – und Sekundenzeiger sind identisch. Sie benutzen lediglich einen anderen Wert als Radius. Alle weiteren Beispielanwendungen sind nach demselben Schema entwickelt und können auf der beiliegenden CD eingesehen werden.
Damit ist mein Ziel erreicht, Echtzeitdaten abzufragen, zu beobachten und visuell darzustellen.

8 Analyse und Performance

Zum Abschluss muss der Server, das Netzwerk und unterschiedliche Typen von Browsern auf ihre Leistungsfähigkeit getestet werden. Ob der Code des Servers eventuell optimiert werden muss lässt sich mit Hilfe von Laufzeitanalysen ermitteln. Damit solche Analysen in der Praxis überhaupt Sinn machen müsste der Server isoliert vom System betrachtet werden. Und gerade das macht aussagekräftige Tests so schwierig. Wie kann also die Prozessorzeit des Servers von der des restlichen Systems abgezogen werden. Noch schwieriger dürfte es sein, die Laufzeit von einzelnen Funktionen und Codeabschnitten zu ermitteln. Es gibt für solche Aufgaben effektive Tools, die extra zur Laufzeit – und Profilanalyse entwickelt worden sind. Unter Linux sind das TIME zur allgemeinen Laufzeitermittlung von Prozessen, der Gnu - Profiler zur Laufzeitermittlung einzelner Funktionen und GCOV, mit dem ermittelt werden kann, wie oft jede einzelne Codezeile innerhalb eines Zyklusses durchlaufen wird. Zur Netzwerk – Analyse wird der Wireshark zum Einsatz kommen, einem Netzwerk – Sniffer, der die genauen Laufzeiten von Daten ermitteln kann. Ob und wie der Browser analysiert werden kann, wird sich zeigen.

8.1 Test des Servers

Damit am Ende der Tests eine verbindliche Aussage getroffen werden kann, werden auf allen Systemen identische Tests in immer gleicher Reihenfolge und mit denselben Werkzeugen durchgeführt. Als Basis sollen Tests auf dem Entwicklungsrechner dienen.

8.1.1 Grobes Zeitverhalten mit TIME

Um sich einen groben Überblick über das Zeitverhalten insgesamt zu verschaffen, eignet sich am besten TIME. TIME liefert drei verschiedene Zeiten mit der folgenden Bedeutung zurück :

real : Die gesamte verstrichene Ausführungszeit ab Ausführung der Kommandozeilen – eingabe
user : Die CPU – Zeit für Benutzerfunktionen; genau die Zeit, welche die Anwendung im User – Level verbracht hat.
sys : Die CPU – Zeit für Systemaufrufe; dies ist die Zeit, welche die Anwendung im Kernel – Level verbracht hat.

Der "real"- Wert ist für Leistungskriterien des Servers ohne Bedeutung, denn er gibt nur an, wie lange der Server lief. Wichtig sind die user und die sys – Zeit.
Die Zeiten beziehen sich auf je 10.000 Anfragen an den Server.

Laufzeittest mit TIME

Serverplattform Linux	accept()	accept + pthread_create	gesamter Request
user	0m0.021s	0m0.143s	0m1.735s
sys	0m0.152s	0m0.542s	0m3.810s

Serverplattform QNX			
user	4,04 sec	4,61 sec	40,46 sec
sys	051 sec	0,80 sec	2,40 sec

Serverplattform RTAI			
user		User – Modus	
sys		identisch mit Linux	

Tabelle 1: gemessenes Zeitverhalten mit TIME

Die gesamte Requestzeit auf einer Linux – Plattform beträgt also 5,545 sec für 10.000 Anfragen. Rund 10% davon (0,542 sec) sind nötig um 10.000 Threads zu erzeugen. QNX benötigt 42,86 sec. um 10.000 Anfragen entgegenzunehmen, was mit der geringeren Taktfrequenz der Plattform zu erklären ist. RTAI ist auf das Linuxsystem nur aufgesetzt. Der Server verhält sich also identisch zum Linuxsystem, solange die Echtzeitanwendungen den User – Modus nicht unterbrechen.

8.1.2 Performance – Test mit gettimeofday(. . .)

Mit TIME kann also relativ genau festgestellt werden, wieviel Prozessorzeit der Server benötigt. Da sich aber der Server den Prozessor mit anderen Programmen teilen muss, geben diese Werte keinen Aufschluss über die tatsächlichen Antwortzeiten.
Um diese zu ermitteln werde ich mit der Funktionen `gettimeofday( timeval *zeit,0)` die genaue Zeit beim Start des Threads und am Ende des Threads ermitteln.Dazu werden folgende Codezeilen in die execute – Funktion eingefügt.

```
void execute( void *arg)
{
   int differenz;
   timeval start, ende;
   gettimeofday( &start, 0);

   /*   Funktionscode   */

   gettimeofday( &ende, 0);
   differenz = ende.tv_usec    start.tv_usec;
   printf(   \n\tDifferenz: %d, differenz);
}
```

Für die Sinuskurve der Beispielseite “03_Seite.html“ wurden bei 10 aufeinander folgenden Anfragen folgende Zeiten gemessen.

Server: QNX – Server
Funktion: Sinus.js
Request: 50 ms

Messung	Start	Ende	Zeit in mikrosekunden
1	835285	837762	2477
2	234912	237076	2164
3	332644	334769	2125
4	406944	409166	2222
5	486909	489268	2359
6	573886	576042	2156
7	654881	657028	2147
8	734918	737157	2239
9	837079	839202	2123
10	914888	917060	2172

Tabelle 2: gemessene Zeiten mit gettimeofday()

Im Durchschnitt liefert der Server nach ca. 2 ms die Antwort. In der Realität kann der Server also nur 500 Antworten/sekunde liefern.
Zum gesamten Request gehört aber auch das Thread erzeugen, Scheduling und Speicherverwaltung. Hierfür werde eine indirekte Messmethode verwenden. Ich werde dazu ein Programm starten, was in einer geschachtelten Schleife Zähloperationen durchführt. Nach einer bestimmten Anzahl von Zähloperationen wird die vergangene Zeit notiert. Im nächsten Schritt wird neben diesem Zählprogramm auch der Server gestartet und Anfragen abarbeiten. Das Zählprogramm verliert dadurch Prozessorzeit an den Server. Die Differenz zur Messung ohne Server ist dann die Prozessorzeit, die der Server für eine bestimmte Anzahl von Anfragen verbraucht.

Das Zählprogramm sieht wie folgt aus:

```
int main( int argc, char **argv)
{
   timeval start, ende;
   int i, j, k, diff_int, diff_mikro;
   gettimeofday( &start, 0);
   for( i = 0; i < 1000; i++)
   {
      for( j = 0; j < 1000; j++)
      {
         for( k = 0; k < 40000; k++)
            ;
      }
   }
   gettimeofday( &ende, 0);
   diff_int = ende.tv_sec - start.tv_sec;
   diff_mikro = ende.tv_usec - start.tv_usec;
   printf( "\nZeit: %d, %d\n\n", diff_int, diff_mikro);
   return 0;
}
```

Wird das Programm 3 – mal gestartet, werden auf meinem Rechner folgende Ausführungszeiten ausgegeben.

Testprogramm

Messung	Zeit
1	73,451794
2	73,398353
3	74,550164

Tabelle 3: Zeitmessung des Testprogramms

Es benötigt also ca. 74 Sekunden als Durchschnitt. Das Sinus – Skript der HTML – Seite wird alle 50 ms die Grafik aktualisieren, was bedeutet, dass in dieser Zeit 1000 Anfragen gestartet werden können. Die Zeit, um die sich das Zählprogramm jetzt verlängert, wird die Zeit sein, die der Server für seine Arbeit benötigt.

Server: QNX – Server
Funktion: Sinus.js
Request: 50 ms
Anzahl der Anfragen: 1000

Messung	Zeit
1	87,343166
2	86,618600
3	87,87398

Tabelle 4: Testprogramm + Server

Die Zeit des Zählprogramms hat sich um rund 13 Sekunden gegenüber der ersten Messung verlängert. D.h., ist der Prozessor hoch belastet, können nur ca. 80 Anfragen/sec. bearbeitet werden. Ob die erstellten RTAI – Anwendungen diese Zeiten erreichen soll ein nächster Test zeigen. Dazu wird die im Anhang beschriebene RTAI – Zeitfunktion `rt_get_time()` verwendet. Die gemessenen Zeiten der RTAI – Uhr lieferten durchweg den Wert 0 ms, da die Zeitauflösung keine genaueren Zeiten zuließ. Es kann also nur gesagt werden, dass die Ausführungszeit bei kleiner einer Millisekunde liegt.

Alle anderen RTAI – Anwendungen werden zeitlich auf keinen Fall über der Zeit der RTAI – Uhr liegen.
Es wird nun unter RTAI derselbe Test durchgeführt wie unter QNX. Es wird dazu wieder die künstliche Belastung (Zählprogramm) gestartet und eine SVG – Füllstandsanzeige über den Browser aufgerufen. Folgende Werte wurden ermittelt:

Server: RTAI – Server
Funktion: fuellstand_blau.js
Request: 50 ms
Anzahl der Anfragen: 1000

Messung	Zeit
1	89,416306
2	89,428857
3	89,330134

Tabelle 5: RTAI - Server im Stress - Test

Bei dieser Messung sind sogar 15 Sekunden nötig, um 1000 Anfragen zu bearbeiten.
Mit dem Netzwerk – Sniffer werden die ermittelten Werte noch einmal überprüft. Da nicht alle 1000 Anfragen ausgewertet werden können, soll jetzt wieder die Uhrzeit für 10 Sekunden gestartet werden.

Für 10 Anfragen sind nachfolgende Werte gemessen worden:

Server: QNX – Server
Funktion: ajaxUhr.html
Messung: 10 sekunden

Request	Nummer	Zeit	Quelle	Ziel	Protokoll	Info
1	4	0,000297	192.168.100.100	141.22.26.34	TCP	3122 → 8082 [PSH,ACK]
	7	0,000450	141.22.26.34	192.168.100.100	TCP	8082 → 3122 [FIN,PSH,ACK]
	Diff	0,000153				
2	14	1,066466	192.168.100.100	141.22.26.34	TCP	3123 → 8082 [PSH,ACK]
	17	1,071845	141.22.26.34	192.168.100.100	TCP	8082 → 3123 [FIN,PSH,ACK]
	Diff	0,005379				
3	24	2,125051	192.168.100.100	141.22.26.34	TCP	3124 → 8082 [PSH,ACK]
	27	2,126595	141.22.26.34	192.168.100.100	TCP	8082 → 3124 [FIN,PSH,ACK]
	Diff	0,001544				
4	34	3,175555	192.168.100.100	141.22.26.34	TCP	3125 → 8082 [PSH,ACK]
	37	3,180986	141.22.26.34	192.168.100.100	TCP	8082 → 3125 [FIN,PSH,ACK]
	Diff	0,005431				
5	44	4,238127	192.168.100.100	141.22.26.34	TCP	3126 → 8082 [PSH,ACK]
	47	4,249251	141.22.26.34	192.168.100.100	TCP	8082 → 3126 [FIN,PSH,ACK]
	Diff	0,036249				
6	54	5,284806	192.168.100.100	141.22.26.34	TCP	3127 → 8082 [PSH,ACK]
	57	5,321055	141.22.26.34	192.168.100.100	TCP	8082 → 3127 [FIN,PSH,ACK]
	Diff	0,011124				
7	64	6,394126	192.168.100.100	141.22.26.34	TCP	3128 → 8082 [PSH,ACK]
	67	6,399553	141.22.26.34	192.168.100.100	TCP	8082 → 3128 [FIN,PSH,ACK]
	Diff	0,005427				
8	74	7,456173	192.168.100.100	141.22.26.34	TCP	3129 → 8082 [PSH,ACK]
	77	7,461832	141.22.26.34	192.168.100.100	TCP	8082 → 3129 [FIN,PSH,ACK]
	Diff	0,005659				
9	84	8,511252	192.168.100.100	141.22.26.34	TCP	3130 → 8082 [PSH,ACK]
	87	8,516660	141.22.26.34	192.168.100.100	TCP	8082 → 3130 [FIN,PSH,ACK]
	Diff	0,005408				
10	94	9,561702	192.168.100.100	141.22.26.34	TCP	3131 → 8082 [PSH,ACK]
	97	9,567053	141.22.26.34	192.168.100.100	TCP	8082 → 3131 [FIN,PSH,ACK]
	Diff	0,005351				

Minimum: 0,000153 sec.
Maximum: 0,036249 sec.
Durchschnitt: 0,008172 sec. rund 8 ms

Tabelle 6: QNX - Server im Stresstest (Wireshark)

Die RTAI – Uhr lieferte mit dem Wireshark folgende 10 Werte:

Server: RTAI – Server
Funktion: uhrzeit.js
Messung: 10 sekunden

Request	Nummer	Zeit	Quelle	Ziel	Protokoll	Info
1	4	0,000302	141.22.26.255	141.22.26.34	TCP	2821 → 8082 [PSH,ACK]
	7	0,001322	141.22.26.34	141.22.26.255	TCP	8082 → 2821 [FIN,PSH,ACK
	Diff	0,00102				
2	14	1,068909	141.22.26.255	141.22.26.34	TCP	2822 → 8082 [PSH,ACK]
	17	1,078551	141.22.26.34	141.22.26.255	TCP	8082 → 2822 [FIN,PSH,ACK
	Diff	0,009642				
3	24	2,135260	141.22.26.255	141.22.26.34	TCP	2823 → 8082 [PSH,ACK]
	27	2,153506	141.22.26.34	141.22.26.255	TCP	8082 → 2823 [FIN,PSH,ACK
	Diff	0,018246				
4	34	3,222544	141.22.26.255	141.22.26.34	TCP	2824 → 8082 [PSH,ACK]
	37	3,229076	141.22.26.34	141.22.26.255	TCP	8082 → 2824 [FIN,PSH,ACK
	Diff	0,006532				
5	44	4,270474	141.22.26.255	141.22.26.34	TCP	2825 → 8082 [PSH,ACK]
	47	4,276845	141.22.26.34	141.22.26.255	TCP	8082 → 2825 [FIN,PSH,ACK
	Diff	0,006371				
6	54	5,335658	141.22.26.255	141.22.26.34	TCP	2826 → 8082 [PSH,ACK]
	57	5,359986	141.22.26.34	141.22.26.255	TCP	8082 → 2826 [FIN,PSH,ACK
	Diff	0,024328				
7	64	6,430615	141.22.26.255	141.22.26.34	TCP	2827 → 8082 [PSH,ACK]
	67	6,437189	141.22.26.34	141.22.26.255	TCP	8082 → 2827 [FIN,PSH,ACK
	Diff	0,006574				
8	74	7,484799	141.22.26.255	141.22.26.34	TCP	2828 → 8082 [PSH,ACK]
	77	7,495417	141.22.26.34	141.22.26.255	TCP	8082 → 2828 [FIN,PSH,ACK
	Diff	0,010618				
9	84	8,546423	141.22.26.255	141.22.26.34	TCP	2829 → 8082 [PSH,ACK]
	87	8,553194	141.22.26.34	141.22.26.255	TCP	8082 → 2829 [FIN,PSH,ACK
	Diff	0,006771				
10	94	9,617436	141.22.26.255	141.22.26.34	TCP	2830 → 8082 [PSH,ACK]
	97	9,623969	141.22.26.34	141.22.26.255	TCP	8082 → 2830 [FIN,PSH,ACK
	Diff	0,006533				

Minimum: 0,001020 sec.
Maximum: 0,024328 sec.
Durchschnitt: 0,009663 sec.

Tabelle 7: RTAI - Server im Stresstest (Wireshark)

8.1.3 Bewertung

Obwohl der Wireshark eine etwas bessere Performance als `gettimeofday()` ermittelt, wird die Messung jedoch bestätigt. Mehr als 100 Anfragen pro Sekunde sind bei Vollast des Zielsystems nicht zu erwarten. Es ist also im Einzelfall zu prüfen, ob diese Rate für eine entsprechende Visualisierung ausreichend ist.

8.2 Client – Test

Die Frage die sich jetzt stellt ist : Ist ein Browser in der Lage eine derartige Anfrageflut überhaupt auszulösen, wenn er noch die Darstellung der gesendeten Daten übernehmen muss. Dazu werden folgende Parametereinstellungen vorgenommen.
Der Request – Intervall wird auf 50 ms eingestellt (bei der Systemzeit nur 1000 ms), da 20 Aktualisierungen pro sec vom Auge noch wahrgenommen werden kann. Gestartet wird der Test mit dem Internet – Explorer von Microsoft. Als Messwerkzeug wird wieder der Wireshark (ein Netzwerk - Sniffer) verwendet.

8.2.1 Test mit Internet Explorer

Als erstes wird die Seite "/Seite_03.html" aufgerufen und die erste Kreisfunktion gestartet. Gemessen wurden im Schnitt Zeiten von 0,08 sekunden (Anfrage – Intervall) für eine Grafik. D.h. die Frequenz von 20 Aktualisierungen/sec erreicht der Browser nicht.
Dabei liegt die Streuung der einzelnen Anfragen zwischen 0,0912 sec und 0,0721 sec, obwohl immer dieselbe Prozedur ausgeführt wird. Ein Grund dafür liegt in der Konstruktion der Browser (gilt für alle Browser). Diese greifen zur Visualisierung nicht auf die Grafikkarte zu, sondern erledigen alles über den Prozessor und bekommen diesen natürlich nicht exklusiv. Für die Uhrzeit ist diese Frequenz natürlich mehr als ausreichend. Die genauen Reaktionszeiten der Uhrzeit sind den Netzwerkprotokollen zu entnehmen.

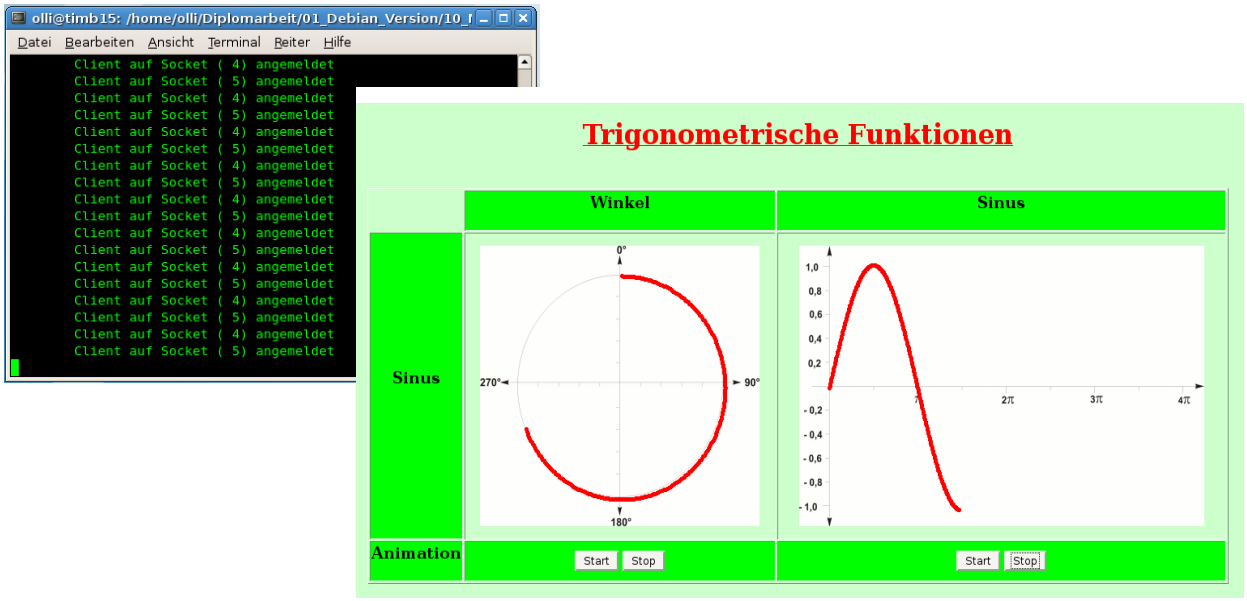


Abbildung 10: Prozessvisualisierung mit Windows - Explorer

Der Internet – Explorer hat aber einen entscheidenden Nachteil. Er kann standardmäßig keine SVG – Grafiken anzeigen. Dazu muss ein spezielles Adobe – Plugin geladen werden. Warum sich Microsoft dieser Technik verweigert war leider nicht herauszufinden.

8.2.2 Test mit Firefox

Für den Test der SVG – Grafiken muss deshalb der Morzilla – Firefox gestartet werden und die Seite "/Seite_02.svg" aufgerufen werden. Die aufwendigste Grafik sollte vom Gefühl her, die Farbverbrauchsanzeige sein. Bei einer eingestellten Requestzeit von 50 ms lagen die gemessenen Zeiten zwischen 0,036 und 0,095 sec. Die Füllstandsanzeigen waren für die Messung auf 100 ms Request – Rate eingestellt worden. Selbst bei 10 Aktualisierungen/sec war der Firefox nicht in der Lage genaue Zeitintervalle einzuhalten. Die Zeiten lagen mit 0,13 und 0,16 sec. (Minimum, Maximum) über dem eingestellten Intervall. Die Uhrzeit lag mit 0,98 sec im Schnitt unter der eingestellten Zeit von 1000 ms und bereitet dem Browser keine Probleme.

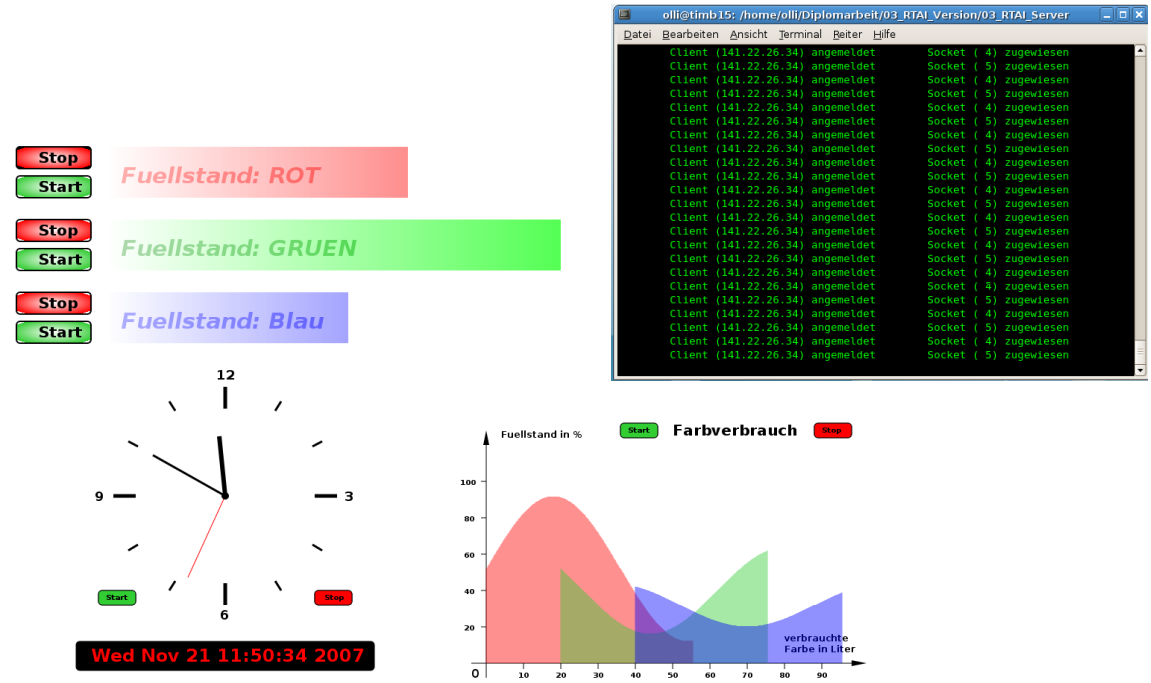


Abbildung 11: Prozessvisualisierung mit Morzilla - Firefox

Dennoch konnte auch der Firefox nicht die gesamte Funktionalität bieten. Bei den Mausbewegungen werden nur Maus – Clicks behandelt. Das Ziehen des Mauszeigers über den Bildschirm wird nicht ausgewertet. Daher zeigt die Farbmischgrafik im rechten oberen Teil des Fensters ein weißes Feld.

8.2.3 Test mit Konqueror

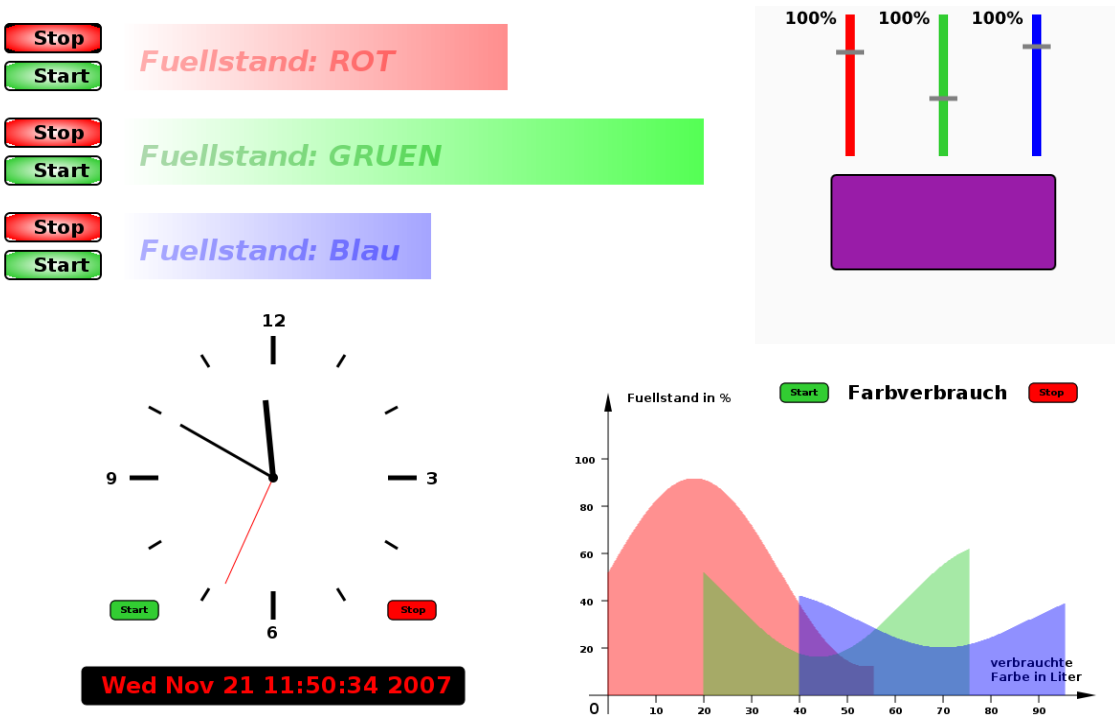


Abbildung 12: Prozessvisualisierung mit Konqueror

Der Konqueror ist der einzige Browser der alle SVG – Grafiken korrekt anzeigen kann. Bei ihm gab es ein anderes Problem. Das Protokoll des Konqueror muss eine andere Struktur haben, als bei allen anderen zum Einsatz gekommenen Webbrowsern. Er schloss clientseitig die Verbindung nach dem Erhalt des Datenpakets. Dadurch blockiert die send() – Funktion, die die Verbindung serverseitig beenden soll. Nach ca. 10 bis 15 Anfragen stürzt der Server dann ab. Wird die Verbindung nicht serverseitig geschlossen (kein Ende Paket “\r\n“) gibt es keine Probleme. Unter Umständen benutzt der Konqueror nicht das HTTP – Protokoll 1.1. Dieses Problem konnte allerdings bis zum Zeitpunkt der Fertigstellung der Arbeit nicht mehr behoben werden.

9 Anhang

Alle restlichen Performance – Tabellen, Erläuterungen und Funktionsbeschreibungen beginnen hier.

9.1 SVG - Elementzugriffe

- ***getElementById()***
  Greift auf das Element mit der entsprechenden Id zu. Beim Anlegen des Elements muss diesem daher eine eindeutige ID zugewiesen werden. Werden Zeichenketten als Parameter übergeben, müssen diese in einfachen oder doppelten Anführungszeichen geschrieben werden.

- ***getElementsByTagName()***
  Greift auf alle Elemente eines Namens zu und liefert eine Liste aller dieser Elemente zurück. Auf die Liste kann jetzt mit getLength() (Länge der Liste) oder getItem() (gib ein bestimmtes Element) zugegriffen werden. Die Nummerierung der einzelnen Elemente beginnt bei 0.

- ***getDocumentElement()***
  Liefert das svg-Element zurück, also das Wurzelelement der Grafik. Diese Methode ist sehr nützlich, wenn Sie Knoten oder Elemente manipulieren wollen.

- ***getAttributes()***
  Diese Methode liefert eine Liste der Attributknoten eines Elementknotens zurück. Auch auf diese Liste kann mit den Funktionen getLength() und getItem() zugegriffen werden.
- ***getAttribute()***
  Diese Methode liefert den Wert eines bestimmten Attributknotens zurück. Dabei wird der Attributname als Parameter übergeben.

- ***setAttribute()***
  Setzt ein Attribut für ein Elementknoten. Erwartet zwei Parameter: den Namen des Attributs (Zeichenkette) und den entsprechenden Wert (Zahl oder Zeichenkette).

- ***removeAttribute()***
  Entfernt ein Attribut aus einem Elementknoten. Erwartet als Parameter den Namen des Attributs (Zeichenkette).

- ***createElement()***
  Erzeugt ein neues Element. Erwartet als Parameter den Namen des Elements.

- ***getData()***
  Diese Methode liefert den Inhalt eines Textknotens zurück.

- ***setData()***
  Mit Hilfe dieser Methode können Sie den Wert eines Textknotens überschreiben. Als Parameter wird die neue Zeichenkette übergeben.

- ***insertData()***
  Durch diese Methode können Sie eine neue Zeichenkette an einer beliebigen Position innerhalb des Textknotens einfügen. Als erster von zwei Parametern wird zuerst die Zeichenpostion angegeben an der die neue Zeichenkette platziert werden soll. Als zweiter Parameter wird die neue Zeichenkette übergeben.

- ***deleteData()***
  Wenn Sie eine beliebige Zeichenkette aus dem Textknoten entfernen möchten, können Sie diese Methode verwenden. Die Methode erwartet zwei Ganzzahlen als Parameter: das Zeichen, ab dem gelöscht werden soll, dabei beginnt die Nummerierung bei 0 und wieviele Zeichen gelöscht werden sollen.

- ***createTextNode()***
  Erzeugt einen Text-Knoten, d.h. die eigentliche Zeichenkette für ein Textelement und nicht das Textelement selbst. Erwartet als Parameter eine Zeichenkette. Derart erzeugte Text-Daten können dann als Kind-Element (d.h. als eigentlicher Text-Inhalt) eines Textelements verwendet werden.

- ***getComputedTextLength()***
  Mit dieser Methode können Sie die Breite eines Textelements, d.h. der gesamten Zeichenkette, in Pixeln ermitteln. Beachten Sie: Diese Methode wird auf den Elementknoten des text-Elements angewandt - und nicht auf den Textknoten.

9.2 SVG Mouse - Events

- ***onclick:*** Klick auf das Element
- ***onactivate:*** Aktivierung des Elements durch beliebigen Zeiger (Mausklick, beliebige Taste usw.)
- ***onmousedown:*** Drücken der linken Maustaste
- ***onmouseup:*** Loslassen der linken Maustaste
- ***onmouseover:*** Den Bereich des Elements mit dem Mauszeiger betreten
- ***onmousemove:*** Bewegen der Maus über dem Bereich des Elements
- ***onmouseout:*** Den Bereich des Elements mit dem Mauszeiger verlassen
- ***onkeypress:*** Taste gedrückt
- ***onkeydown:*** Taste gedrückt
- ***onkeyup:*** Taste losgelassen
- ***onfocusin:*** Das Element bekommt den Fokus
- ***onfocusout:*** Das Element verliert den Fokus

9.3 RTAI - API

9.3.1 RTAI – Taskfunktionen

```
RT_TASK task;
int rt_task_init( &task, void(*rt_thread)(int), int daten,
                  int stack_size, int prioritaet, int uses_fpu,
                  void(*signal)(void));
```

Beschreibung: Die Funktion erzeugt einen Echtzeit – Task. `task` ist ein Zeiger auf den erzeugten Prozess, der dem Programm zur Verfügung gestellt wird.
Wichtig ist der reservierte Stackspeicher in Bytes; der sollte nicht zu knapp bemessen sein. Alle anderen Parameter sind optional und können gegebenenfalls mit NULL besetzt werden. Bei der Priorität kann NULL einen ungültigen Wert darstellen und einen Compilerfehler zur Folge haben.
Rückgabe: bei Erfolg 0, bei Misserfolg ein negativer Wert

```
RT_TASK task;
int rt_task_delete( &task);
```

Beschreibung: löscht einen vorher mit rt_task_init erzeugten Task.
task zeigt auf eine Taskstruktur
Rückgabe: bei Erfolg 0, ansonsten einen negativer Wert

```
RT_TASK task;
int rt_task_make_periodic( &task, RTIME start_time, RTIME periode);
```

Beschreibung: Die Funktion startet einen periodischen Prozess im Intervall periode. Der Start beginnt zur Startzeit start_time.
Rückgabe: bei Erfolg 0, ansonsten einen negativen Wert

```
RT_TASK task;
int rt_task_suspend( &task);
```

Beschreibung: suspend stoppt die Ausführung eines Prozesses, beendet ihn aber nicht. Der Prozess verbleibt im Scheduler.
Rückgabe: bei Erfolg 0, ansonsten einen negativen Wert

```
RT_TASK task;
int rt_task_resume( &task);
```

Beschreibung: Mit resume kann der zuvor suspendierte Prozess wieder gestartet werden.
Rückgabe: bei Erfolg 0, ansonsten einen negativen Wert

```
void rt_busy_sleep( int nanosecs);
void rt_sleep( RTIME delay);
void rt_sleep_until( RTIME time);
```

Beschreibung: `busy_sleep` verzögert die Ausführung des Prozesses um die Zeit, die als Parameter übergeben wird. Die Kontrolle kehrt aber nicht zum Scheduler zurück. D.h., der Scheduler kann diese Zeit nicht verplanen und die CPU an andere wartende Tasks vergeben. Bei `rt_sleep` passiert genau das. Die Ausführung des Task wird nach hinten verschoben. Er wird im Scheduler an die Stelle eingereit, an der sein delay abläuft. `sleep_util` funktioniert ähnlich, lässt sich aber für längere Zeiten besser handhaben. Hier kann ein Prozess für lange Zeit aus dem Schedulerbetrieb ausgeschlossen werden.
Rückgabe: keine

9.3.2 RTAI – Zeitfunktionen

```
RTIME start_rt_timer( int periode);
```
Beschreibung: Startet einen Timer zu dem Zeitpunkt, an dem das Programm die Anweisung erreicht. Er wird dann immer wieder gestartet, mit der Periode, die als Parameter übergeben wurde.
Rückgabe: Die Periode in Timer – Ticks

```
RTIME rt_get_timer();
```
Beschreibung: Gibt die Anzahl der Realtime – Ticks pro Sekunde zurück, die abgelaufen sind, seitdem das Programm gestartet wurde.
Rückgabe: Zeit in Timer – Ticks

```
RTIME rt_get_time_ns();
```
Beschreibung: Gibt die Zeit in Nanosekunden zurück, die seit dem Programmstart vergangen sind.
Rückgabe: Zeit in Nanosekunden

```
int rt_task_make_periodic( &task, RTIME start_time, RTIME periode);
```
- Der erste Parameter steht für den zu steuernden Prozess.
- Danach folgt der **absolute** Zeitpunkt der ersten Ausführung.
- Der letzte Wert beschreibt den Zeitabstand zwischen allen weiteren Aktivierungen.

Am Ende eines Prozesses sollten im cleanup – Teil die Timer wieder gestoppt werden, denn nur gestoppte und dadurch wieder freigegebene Timer können neu vergeben werden.

```
void rt_stop_timer();
```
Beschreibung: Stoppt den vorher gestarteten Timer wieder.
Rückgabe: keine

Zwei ganz nützliche Ergänzungsfunktionen sollte nicht unerwähnt bleiben.
```
RTIME nano2count( RTIME t);
RTIME count2nano( RTIME t);
```

Beschreibung: Diese Funktionen rechnen Timer – Ticks in Nanosekunden um und umgekehrt.
Rückgabe: der entsprechende RTIME – Zeitwert

9.3.3 FIFO – Kommunikation Linux

```
# include <sys/types.h>
# include <sys/stat.h>
int mkfifo( const char* pfadname, mode_t modus);
```
Beschreibung: Anlegen eines Fifo in einem bestimmtem Pfad (Rendezvous – Prinzip). Als Argument stehen `O_RDONLY`, `O_WRONLY` und `O_RDWR` zur Verfügung. Diese Konstanten können allerdings nicht `ODER` – verknüpft werden.
Mit anderen Flags sind sie allerdings kombinierbar. Diese wären `O_CREAT`, `O_APPEND`, `O_EXCL`, `O_TRUNC`, `O_SYNC`, `O_NONBLOCK` und `O_NOCTTY`. Auch sämtliche Zugriffsrechte – Makros wie `S_IRUSR`, `S_IWUSR` usw. sind zulässig.
Rückgabe: Fifo erfolgreich angelegt = 0;
ansonsten -1

Bearbeitet werden kann das FIFO mit den normalen Operationen, die auch für Dateien gültig sind.

```
int open( const char *fifo_name, int flags, mode_t zugriffsrechte);
```
Beschreibung: Öffnet einen zuvor angelegten FIFO. Das Argument `zugriffsrechte` ist optional und kann weggelassen werden.
Rückgabe : Bei Erfolg wird die FIFO – Nummer zurück geliefert
ansonsten -1

```
ssize_t write( int fifo, void *puffer, size_t anzahl_bytes);
```
Beschreibung: Mit `write()` werden `anzahl_bytes` Zeichen, ab Adresse `puffer` in den Fifo geschrieben.
Rückgabe: Bei Erfolg die Anzahl der erfolgreich geschriebenen Zeichen
ansonsten -1

```
ssize_t read( int fifo, void *puffer, size_t anzahl_bytes);
```
Beschreibung: Mit `read()` werden anzahl_bytes aus einem geöffneten FIFO in die Adresse `puffer` kopiert.
Rückgabe: Anzahl der erfolgreich gelesenen Bytes
ansonsten -1

9.3.4 FIFO – Kommunikation RTAI

```
# include <rtai_fifos.h>
int rtf_create( unsigned int fifo, int groesse);
```
Beschreibung: Diese Funktion erzeugt eine Realtime – FIFO mit der angegebenen groesse in Bytes und der Fifonummer. Alle FIFOs sind in dieser Form[30] in `/dev/rtfX` zu finden, wobei X für eine Zahl zwischen 0 und 63 steht.
Rückgabe: Fifo erfolgreich angelegt 0
ansonsten negativen Wert.

```
int rtf_destroy( unsigned int fifo);
```
Beschreibung: Diese Funktion schließt den FIFO wieder, der zuvor erzeugt wurde.
Rückgabe: FIFO erfolgreich freigegeben 0
ansonsten negativen Wert.

```
int rtf_reset( unsigned int fifo);
```
Beschreibung: Diese Funktion setzt den FIFO zurück. Er befindet sich danach in dem Zustand, als wäre er gerade erzeugt worden. Auf gegebenenfalls vorhandene Daten im Fifo kann nicht mehr zugegriffen werden, da der Referenzzeiger des Speicherbereichs auf 0 gesetzt wird.
Rückgabe: Fifo erfolgreich freigegeben 0
ansonsten negativen Wert.

```
int rtf_resize( unsigned int fifo, int groesse);
```
Beschreibung: Diese Funktion definiert die Größe des FIFO im laufenden Programm neu. Noch im FIFO befindliche Daten gehen dabei verloren.
Rückgabe: Größe des FIFO s in Bytes.
ansonsten negativen Wert.

30 Das Präfix **`rtf`** steht für Realtime – FIFO

```
int rtf_put( unsigned int fifo, void *puffer, int groesse);
```

Beschreibung: Die Funktion schreibt einen Datenblock in den vorher erzeugten FIFO. `fifo` ist die id, also der Rückgabewert von `rtf_create()`. `puffer` ist der Datenblock der in den FIFO geschrieben werden soll und mit `groesse` ist die Größe des `puffer` – Bereichs in Byte gemeint.

Rückgabe: Anzahl der geschriebenen Bytes. Sie muss mindestens um 1 kleiner sein als die Größe des Puffers.
ansonsten negativer Wert.

```
int rtf_get( unsigned int fifo, void *puffer, int groesse);
```

Beschreibung: `rtf_get()` liest aus einem FIFO. Mit `groesse` muss die Anzahl der Bytes angegeben werden die man erwartet, damit Speicher bereit gestellt werden kann.

Rückgabe: Anzahl der gelesenen Bytes
ansonsten negativer Wert.

9.3.5 FIFO – Handler RTAI

```
int rtf_create_handler( unsigned int handler,
                        int (*handler)( unsigned int fifo));
```

Beschreibung: Die Funktion richtet einen Handler ein, der ausgeführt wird, wenn Daten aus dem FIFO gelesen werden können oder wenn in ihn hineingeschrieben wurde. Die `handler` können durchnummeriert werden, sind aber mit der Funktion als zweiten Parameter fest verbunden. Sie können dann, wie in der Abbildung 5 zusehen, dem Absender auf demselben FIFO antworten. Er muss aber nicht antworten, es wird nichts blockiert.

Rückgabe: Bei Erfolg 0
ansonsten negativer Wert.

Stichwortverzeichnis

[1]	*www.ea-online.de/ea/live/fachartikelarchiv/ha_artikel/detail/*		Nov 2007
[2]	*www.gefanuc.com/*		Nov 2007
[3]	*www.automation.siemens.com/hmi/html_00/products/software/wincc_optionen/navigator.htm*		Nov 2007
[4]	*www.doebelt.de/produkte/neu/webnet.html*		Nov 2007
[5]	*http://de.wikipedia.org/wiki/Webserver*		Nov 2007
[6]	*www.hughes.com.au/products/libhttpd/*		Nov 2007
[7]	Jürgen Wolf,	Linux – Unix – Programmierung Gallileo Press	2005
[8]	Johannes Gamperl,	Ajax und Web 2.0, Gallileo Press	2006
[9]	Johann-Christian Hanke,	Websites,	2003
[10]	http://docs.mootools.net/		Nov 2007
[11]	J. David Eisenberg	SVG Essentials, O'Railly,	2002

ergänzende Informationen

http://www.fh-wedel.de/~si/praktika/MultimediaProjekte/SVG/SVG_Tutorial_mi3794/2_6_2.htm
http://svg.tutorial.aptico.de/
http://svglbc.datenverdrahten.de/
http://schoen.priv.at/pc104.html
http://www.fbmnd.fh-frankfurt.de/~doeben/I-RT/I-RT-TH/VL6/i-rt-vl-vl6.html#rtai
http://www.nwlab.net/tutorials/ethereal/ethereal-tutorial.html
http://www.easy-network.de/ethereal.html
http://wd-testnet.world-direct.at/mozilla/dhtml/funo/jsTimeTest.htm
http://www.computerbase.de/news/software/browser/2006/august/opera_performance-koenig_javascript/
http://de.wikipedia.org/wiki/Java-Applet

Inhalt der CD

1. Eine Linux – Version des Servers mit den Beispielanwendungen /03_Seite.html und /ajaxUhr.html
2. Eine QNX – Version des Servers mit den Beispielanwendungen /03_Seite.html und /ajaxUhr.html
3. Eine QNX – Version eines seriell arbeitenden Servers mit den Beispielanwendungen /03_Seite.html und /ajaxUhr.html
4. Eine RTAI – Version des Servers mit der Beispielanwendung /Seite_02.svg
5. Die Bachelor – Arbeit im PDF – Format

Versicherung der 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 den _____ _____

Ort, Datum Unterschrift