Projektbericht
Theater der Maschinen

von

Khaled Abdelrahimsai-Pjau, Vahit Bilmez, Denis Böhme, Frank Euhus,
Marcus Fährer, Torsten Fröhling, Thomas Gnewuch, Mathias Liebert,
Daniel Mutis, Jörn Raffel, Ersin Ürer, Oliver Weickmann

Ein studentisches Projekt an der Universität Bremen
vom Wintersemester 1998/1999 bis Sommersemenster 2000

Betreut von

Prof. Dr. Wilhelm Bruns
und
Dipl.-Inf. Bernd Robben

15. Januar 2002

Dies ist der Abschlussbericht des Projektes *Theater der Maschinen*. Dieses Projekt ist ein studentisches Projekt, das im Hauptstudium an der Universität Bremen absolviert wird, d.h. dass ein Projekt im Hauptstudium zu belegen ist.
Dieses Projekt wurde von Professor Wilhelm Bruns im Somersemester 1998 vorgeschlagen, begann dann im Wintersemester 98/99 und endete im Sommersemester 2000. Es wurde dabei von Professor Wilhelm Bruns und vom Diplominformatiker Bernd Robben betreut.
Es ist das wichtigste Teil im Hauptstudium in Bremen und wird von einigen Veranstaltungen begleitet, die im nächsten Kapitel beschrieben werden. Darüberhinaus sollen wir Studierende dabei Projektarbeit als solche, mit der wir im späterem Leben wohl noch viel zu tun haben werden, aber ebenso die konkrete Praxis von Projektarbeit kennenlernen.
Es geht dabei darum, schon im Studium die Arbeit innerhalb eines Projektes (einer Gruppe) kennenzulernen und selbstständig zu koordinieren. Die Ergebnisse des Projektes werden dann im Rahmen des Projekttages im Fachbereich allen Interessierten dargeboten, in Form einer Art *Performance* und durch eine Präsentation durch Plakate/Schauwände.
Ebenso gehört zum Projekt ein Abschlussbericht, der nun hier vorliegt, und in dem alle wichtigen Aspekte noch einmal erläutert werden, insbesondere die technischen Einzelheiten, die naturgemäß in den Vorführungen zu kurz kommen müssen. Außerdem gehören auch die Reflexionen der gesamten Gruppe aber auch jeder einzelnen Person zum Projekt und der Projektarbeit dazu.

Dieser Bericht ist so aufgebaut, dass zunächst das Ergebnis und der Weg dorthin vorgestellt werden sowie die zugehörigem Aktivitäten und Veranstaltungen. Danach kommt der technische Teil, in dem alle Unterprojekte sich selber beschreiben. Im anschließenden Kapitel folgt die Schilderung der *künstlerischen Umsetzung* der Projektidee.
Zum Schluss folgt das Gesamtfazit, gefolgt von den persönlichen Fazits der Projektteilnehmer.

Inhaltsverzeichnis

1 Einführung **11**
- 1.1 Themenfindung . . . 11
- 1.2 Begleitende Veranstaltungen . . . 12
- 1.3 kultureller Teil . . . 13
- 1.4 Das Ergebnis . . . 13

2 Mülleimer **15**
- 2.1 Einleitung . . . 15
- 2.2 Bierkiste . . . 15
- 2.3 Luftpuppe . . . 16
- 2.4 Mülleimer . . . 16
- 2.5 Aufbau des Mülleimers . . . 16
- 2.6 Mikrocontroller . . . 18
- 2.7 Schrittmotoren . . . 20
- 2.8 Relaissteuerung . . . 22
- 2.9 Joysticksteuerung . . . 25
- 2.10 Computersteuerung . . . 25

3 Master Control Program **29**
- 3.1 Entstehung . . . 29
- 3.2 Prinzip des MCP . . . 29
- 3.3 Fazit . . . 32

4 Der SoundRaum **33**
- 4.1 Ideenfindung . . . 33
- 4.2 Verschiedene Abstrahlungstechniken für Musik . . . 33
  - 4.2.1 Mono . . . 33
  - 4.2.2 Stereo . . . 34
  - 4.2.3 Dolby Surround . . . 35
  - 4.2.4 Konzept des 3D-Sounds . . . 35
- 4.3 SoundRaum Inside . . . 36
  - 4.3.1 Verwendete Produkte . . . 37
- 4.4 Die SoundControl DLL . . . 40
- 4.5 Der SoundServer . . . 42
- 4.6 Die Sound Description Language . . . 45
  - 4.6.1 Beispiele . . . 45
- 4.7 SNALG . . . 47
  - 4.7.1 Funktionsweise und Beschreibung . . . 47
  - 4.7.2 Aufgetretene Probleme . . . 48
- 4.8 Fazit . . . 48

5 Lichtsteuerung **51**
5.1 Einleitung . . . 51
5.2 Aufgabenstellung . . . 51
5.3 Relaiskarte . . . 52
5.4 Server-Software . . . 54
5.5 Client-Software . . . 54
5.5.1 Systemumgebung . . . 54
5.5.2 Benutzer/ Bediener . . . 54
5.5.3 Anforderungen . . . 54
5.5.4 Plattform- und Standortunabhängigkeit . . . 55
5.5.5 Systemspezifikation . . . 56

6 Fliwatüt **59**
6.1 Allgemeiner Teil . . . 59
6.1.1 Motivation . . . 59
6.1.2 Hardware - Überblick . . . 64
6.1.3 Software - Überblick . . . 65
6.2 Spezieller Teil . . . 66
6.2.1 Hardware . . . 66
6.2.2 Softwareentwicklung AVR . . . 74
6.2.3 Software PC . . . 79
6.2.4 Software-Entwicklung PC . . . 79
6.3 Probleme bei der Entwicklung . . . 81
6.4 Anhang . . . 82

7 Marion **87**
7.1 Einleitung . . . 87
7.2 Hardware . . . 87
7.3 Betriebssystem . . . 88
7.4 Marion - Die mechanische Marionette . . . 89
7.4.1 Übersicht . . . 89
7.4.2 Mechanik . . . 89
7.4.3 Elektronik . . . 90
7.4.4 Diskussion . . . 91
7.5 Der Galgen - Positionsbestimmung im Raum . . . 91
7.5.1 Übersicht . . . 91
7.5.2 Mechanik . . . 92
7.5.3 Elektronik . . . 92
7.5.4 Software . . . 94
7.5.5 Diskussion . . . 95
7.6 Hwserver . . . 96
7.6.1 Anforderungen an den Hwserver . . . 96
7.6.2 Entwicklung, Entwurf und Aufbau des Hwservers . . . 97
7.6.3 Schnittstellen des Hwserver . . . 99
7.6.4 Protokoll des Hwservers . . . 100
7.6.5 Implementierung . . . 100
7.6.6 Diskussion . . . 101
7.7 Client zum Hwserver . . . 101
7.7.1 C-Module zur Verwendung in Clients . . . 101
7.8 MarioSeq . . . 101
7.8.1 Anforderungen . . . 102
7.8.2 Entwurf und Entwicklung . . . 102

7.8.3 Implementierung . . . 102
7.8.4 Schnittstellen . . . 103
7.8.5 Eigene Bewertung und Schlussfolgerung . . . 105
7.8.6 Verwendete Tools und Bibliotheken . . . 105

8 Von der Wurst zum Kubismus 107
8.1 Einleitung . . . 107
8.2 Würstchenbude . . . 107
8.2.1 Grundidee . . . 107
8.2.2 Anforderungsdefinition . . . 107
8.2.3 Vorgehensweise . . . 108
8.2.4 Komponenten der Würstchenbude . . . 108
8.2.5 Resultat . . . 109
8.3 VRML-Marionette . . . 109
8.3.1 Anforderungsdefinition . . . 109
8.3.2 Machbarkeitsanalyse . . . 110
8.3.3 VRML . . . 110
8.3.4 Technische Ausstattung . . . 110
8.3.5 Hardware . . . 111
8.3.6 Software . . . 111
8.3.7 Betriebssystem . . . 111
8.3.8 Entwicklungs-Tools . . . 111
8.3.9 Protokolle . . . 111
8.3.10 Resultat . . . 111
8.4 Java3D-Marionette . . . 112
8.4.1 Technische Ausstattung . . . 112
8.4.2 Entwicklung, Entwurf und Aufbau des 3D-Männleins . . . 113
8.4.3 Ein- und Ausgabegeräte . . . 115
8.4.4 Eingabegeräte . . . 116
8.4.5 Ausgabegeräte . . . 116
8.4.6 Das Starten des Programms . . . 116
8.4.7 Resultat . . . 116
8.4.8 Kinematik . . . 117
8.5 Vorlesungen, Seminare und Workshops . . . 118
8.6 Schlusswort . . . 118

9 Die Präsentation 121
9.1 Das inhaltliche Konzept . . . 121
9.2 Die Bühne . . . 121
9.3 Proben und Improvisation . . . 122
9.4 Das Skript . . . 122
9.5 Werbung: Programmheft und Plakat . . . 123
9.6 Die Projekttagpräsentation . . . 123
9.7 Die Aufführung in Stuttgart . . . 124

10 Fazits 125
10.1 Gesamtfazit . . . 125
10.2 Persönliche Fazits . . . 127

Abbildungsverzeichnis

2.1 Der Mülleimer ohne Verkleidung . . . 19
2.2 Der Mikrocontroller . . . 20
2.3 Die Blackbox des Mikrocontrollers . . . 21
2.4 Schrittmotor mit Getriebe und Rad . . . 22
2.5 Platine der Relaissteuerung . . . 23
2.6 Schaltbild zur Realaisteuerung . . . 24
2.7 Die Blackbox der Relaissteuerung . . . 24
2.8 Die Handsteuerung . . . 26
2.9 Grafische Oberfläche der Echtzeit-Steuerung . . . 27
2.10 Grafische Oberfläche des Sequencers . . . 27

3.1 Struktur des Master Control Programmes . . . 30
3.2 Screenshot vom MCP bei der Arbeit . . . 31
3.3 Die Grammatik für den MCP Ablauf . . . 32

4.1 Mono Sound . . . 34
4.2 Stereo Sound . . . 35
4.3 Dolby Surround . . . 36
4.4 3D Sound . . . 37
4.5 Komponenten des SoundRaumes . . . 38
4.6 Die vordefinierten EAX Umgebungseffekte . . . 39
4.7 Klassenstruktur innerhalb von SoundControl . . . 41
4.8 Interne Datenstrukturen . . . 42
4.9 Die nähere Beschreibung der Attribute von SOUNDINFO . . . 43
4.10 CORBA Schnittstelle zum SoundServer . . . 44
4.11 Die formale Grammatik für die Sound Description Language . . . 46

5.1 Die umgebaute Relais-/Opto-Koppler-Karte . . . 53
5.2 Eine schematische Darstellung der Relais/Opto-Koppler-Karte . . . 54
5.3 Zusammenspiel des Servers, Clients und der Relaiskarte . . . 55
5.4 Durch anklicken der Lichtsteuerung wird unsere GUI gestartet . . . 55
5.5 Die Oberflaeche ist dargestellt . . . 56
5.6 Durch Angabe des Hosts und des Ports kann man sich problemlos mit dem Server verbinden . . . 56
5.7 Editor mit dem man verschiedene Sequenzen erstellt, die anschliessend auf der GUI sichtbar werden . . . 57
5.8 Die erstellten Sequenzen sind auf der GUI sichtbar . . . 57
5.9 Protokolldatei . . . 58
5.10 Auf der GUI erkennt man, dass die gelben Lichter an sind und die grauen Lichter aus. Man kann auch direkt von der Oberfläche Lichter an- und ausschalten, sowie sich mit dem Server verbinden und trennen. Die Oberfläche ist direkt manipulierbar 58

6.1 Beispiel . . . 66

6.2 Schema AVR 67
6.3 Schema Bumper 68
6.4 Stifthalterung 69
6.5 Antriebsmechanik Fliwatüt 69
6.6 Schema EDE 1204 IC 70
6.7 Aufbau Servo-Platine 71
6.8 Kaskadierung der Timer 76
6.9 Odometrie-Berechnung 80
6.10 Fliwatüt bei den Proben - Seitenansicht 82
6.11 Fliwatüt bei den Proben - Seitenansicht der Stifthalterung 82
6.12 Draufsicht 83
6.13 Aufbau und Verkabelung der Elektronik 83
6.14 Detailansicht: Verkabelung der Schrittmotorsteuerung 83
6.15 Detailansicht: Aufbau der Schritt- und Servomotorsteuerung 84
6.16 Spannungsquelle der Steuerung und AVR Prozessor 84
6.17 Detailansicht: Stiftaufhängung 84
6.18 Draufsicht 85
6.19 Fliwatüt während der Fahrt auf der Bühne 85

7.1 Marion - Der Körper 89
7.2 Marion - Die Freiheitsgrade und ihre Abhängigkeiten 90
7.3 Marion - Die Mechanik 91
7.4 Marion - Die Mechanik im Detail 92
7.5 Die Aufnahme der Arm-, Bein- und Rumpfmechanik. Sie beinhaltet die Servos für die Drehung von Rumpf/Beine und Arme. 93
7.6 Die Mechanik zur Ansteuerung der Beinehöhen und der Rumpfbeugung. Die Zeichnung zeigt die Mechanik von vorne gesehen. Oberhalb der Bein-Servos befinden sich die Umlenkungen für die Bein-Fäden. Die Umlenkungen werden auf Abstand gehalten, damit sich die Fäden nicht in der Mechanik verhäddern. . . . 94
7.7 Die Mechanik zur Ansteuerung der Arme von vorne gesehen. 95
7.8 Marion - Das Getriebe 96
7.9 Eingabegerät: Das Kreuz 96
7.10 Schaltplan: Drucksensor 97
7.11 Spannungsteiler für die Drucksensor 97
7.12 Spannungsteiler für die Drehungwinkel 98
7.13 Schichten des Hwservers 98
7.14 Der Sequenzer mit Gtk+-Benutzerinterface 104
7.15 Gesamtarchitektur 106

8.1 Würstchenbude 108
8.2 Simulation des VRML-Männleins im Cosmoplayer unter Netscape 110
8.3 Java3d-Männlein-Architektur 114
8.4 3D-Männlein in einem Java Appletviewer 115
8.5 Aufführung des Java 3D-Männleins in einer Interaktion mit der realen Marionette während der Projektpräsentation 117

Tabellenverzeichnis

6.1 Steckerbelegung Motor-Driver IC - Stepper-Motor 67
6.2 Pinbelegung Stecker Motor-Driver IC - Atmel AVR 67
6.3 Geschwindigkeiten im FullStep-Modus 68
6.4 Geschwindigkeiten im HalfStep-Modus 68
6.5 Kommandos in der Setup-Betriebsart 72
6.6 Urpsrungswerte in der Setup-Betriebsart 73
6.7 Kommandos in der Aktiven-Betriebsart 74

Kapitel 1

Einführung

Das Projekt *Theater der Maschinen* begann mit der eher indifferenten Vorstellung der Idee, die Willi Bruns hatte, als er ein Video von Chico MacMurtrie sah, in dem einfach gebastelte Roboter so menschlich agierten, dass man ihnen menschliche Regungen (Gefühle?) zuspricht.
Die grobe Idee bestand also darin, ein Maschinenpark zu entwerfen, der Theater spielt ebenso wie diese Roboter. Der technische und künstlerische Hintergrund blieben bewusst unberücksichtigt. All dies war als offenes, freies und selbstorganisiertes Projekt geplant. Diese Offenheit bezog sich nicht nur auf die Gestaltung, sondern auch auf die Organisation des Projektes, die wir selbst zu leisten hatten. Im Gegensatz zu anderen Projekten waren wir also in Form und Thema weitgehend selbstorganisiert, was auch Einfluss auf die TeilnehmerInnen hatte. Es gab vergleichsweise viele ältere Studierende (auch mit erfolgreichem Abschluss), die uns dann leider nach besseren Angeboten verliessen, auch damit muss man die hohe Verlustquote an ProjektteilnehmerInnen erklären.
In diesem Kapitel wird nun die grobe Konzeptidee beschrieben, ihre Entstehung, und wie sie langsam verändert wurde. Ebenso geht es um die Veranstaltungen, die dieses Projekt begleiteten, und weiteren Aktivitäten, die wir gemeinsam unternommen haben, wie unsere vielfältigen Projektfahrten oder der Besuch anderer *kultureller* Verantaltungen zur Inspiration.

1.1 Themenfindung

Zu Beginn des Projektes gab es nur die vage Themenstellung des Namens, das heisst Maschinen zu entwickeln und mit ihnen etwas auf eine Bühne zu bringen. Was dies für Maschinen sein sollten und welch ein Stück sie in welcher Art darstellen sollten, war allen unklar. So begannen wir auch eher mit der technischen Seite, also mit verschiedenen Möglichkeiten Maschinen zu bauen und auch zu steuern, schnell war auch klar, dass diese auch *irgendwie* miteinander kommunizieren sollten.
Der erste Einschnitt war die erste Projektfahrt, die neben dem Kennenlernen das Ziel hatte, eine Projektidee zu entwickeln. In kleinen Gruppen wurde erst diskutiert, was man machen könnte und wie sich einzelne dort und im Gesamtprojekt einbringen. Diese Ergebnisse der Kleingruppen wurden dann im Plenum vorgestellt und diskutiert. Es setzte sich die Idee durch *etwas krankes* zu machen, in der Art, dass es eine Art Raum wird, in dem sich verschiedene Objekte befinden, die langsam ein Eigenleben entwickeln und aus dem Normalen etwas bedrohlich unheimliches, feindliches machen. Der Raum selber sollte eine Art begehbare Installation werden, wir wollten auf keinen Fall irgendwie als Schauspieler agieren. Dies führte dann zu der Idee in kleinen Gruppen jeweils kleine Objekte, die sich in dem Raum befinden zu entwickeln. Das haben wir auch im zweiten Semester getan, aber schon bald war klar, dass mit den wenigen Dingen und ihren Möglichkeiten die Ursprungsidee nicht zu verwirklichen war.
Wir hatten also unsere Objekte, aber keinen Rahmen mehr dafür. Auf der nächsten Projektfahrt sollte dazu dann ein *Dach* gefunden werden. Nun wurde aus der begehbaren Installation schon eine Bühne und die bereits zum Teil entwickelte Marionette wurde zur Hauptfigur.

Im weiteren begann die Zusammenarbeit mit Jörg Richard, der sehr an unserem Projekt interessiert war. Er stellte uns dabei vor, was es in verschiedenen Zeiten schon für Maschinen im Theater gegeben hat, sozusagen als Inspiration.
Er war es auch, der das Thema Mensch-Maschine-Interaktion zum Inhalt des Stückes machte und die Zusammenarbeit mit Marc, einem Cellisten der Bremer Kammerphilharmonie, initiierte. Somit hatten wir doch einen Darsteller.
Im Laufe der Proben begannen wir erst zu improvisieren, ohne genaue inhaltliche Vorgaben. Dann aber übernahm Jörg Richard mit einem Skript, einem schon fast vollständig ausgearbeiteten Stück, die gesamte künstlerische Leitung. Nun hatten wir also ein Theaterstück und einen Inhalt dazu, nämlich das Thema Mensch-Maschine-Natur, außerdem gab es jetzt doch noch einen Darsteller aus unserer Runde, nämlich Frank.
So kamen wir doch insgesamt zu einem Theater der Maschinen, d.h. ein Stück mit von uns entworfenen Maschinen, aber von der Ursprungsidee, die mal von uns entwickelt wurde, haben letztendlich auch nur diese Maschinen überlebt. Nichtsdestoweniger war es immer unsere gemeinsame Idee, nicht eine, die von außen kam, sondern wir konnten uns immer mit unserem Projekt identifizieren.

1.2 Begleitende Veranstaltungen

Im Laufe der vier Semester gab es neben dem wöchentlichen Projektplenum, das vor allem organisatorischen Fragen gewidmet war, auch inhaltliche Veranstaltungen verschiedenster Art und Form, die zum Gesamtprojekt gehörten.
Im ersten Semester war ein Teil des Projektplenums dem Thema Sensoren und Aktoren gewidmet, die wir uns gegenseitig durch Kurzreferate vorgestellt haben. Ziel war es, technische Möglichkeiten zur Steuerung der zu entwerfenden Maschinen kennenzulernen.
Außerdem gab es die Veranstaltung von Bernd Robben **Sinnenspiel der Medien**. Thema der Veranstaltung war die Änderung der Wahrnehmung durch den Einsatz neuerer Medien, also z.B. die Ablösung der Sprache durch die Schrift. Hierzu gab es auch einen Seminarteil, in dem Referate zu bestimmten Themen gehalten wurden, die dann auf eine multimediale CD geb(r)annt wurden.
Eine weitere Veranstaltung war ein Pneumatikkurs an zwei Nachmittagen, den Hermann Gathmann an der Schule im Holter Feld für uns durchführte, damit wir praktische Erfahrungen zur Steuerung durch Pneumatik sammeln konnten. Dieses Gebiet war für alle neu und hat viel Spaß gebracht.
Eine weitere Möglichkeit, Maschinen zu entwerfen, bot sich mit dem neuen Programm der Firma Lego, LegoMindstorms. Es besteht auf der Basis herkömmlicher Legobausteinen aus Motoren und Sensoren und einer programmierbaren Zentraleinheit zur Steuerung. Auch damit haben wir experimentiert, ohne es später im Projekt einzusetzen.

Im zweiten Semester gab es die Veranstaltung **Modellierung und Simulation** von Dieter Müller. In dieser Veranstaltung lernten wir einige konkrete Simulationsprogramme kennen, es ging hierbei um den konkreten Praxiseinsatz bereits existierender Programme.
Im Projektplenum erzählte Jörg Richard, ein Professor aus dem Studiengang Kulturwissenschaften etwas über den Einsatz von Maschinen im Theater in verschiedenen Epochen. Diese Vorträge waren eher unregelmäßig, führten dann aber zu seiner Veranstaltung im anschließenden Semester. Ziel war es, erst einmal einen Überblick zu bekommen und eventuell Inspirationen zu sammeln.
Außerdem veranstaltete das artec eine selbstorganisierten JAVA-3D-Kurs, an dem auch zwei Leute von uns sehr viel für ihre Programmierung gelernt haben.

Im dritten Semester gab es die Fortstzungsveranstaltung von Jörg Richard **Automaten, Androiden und Roboter**, die im Studiengang Kultuwissenschaften stattfand und in Form von

Referaten einzelne Themen wie Marionettentheater oder den Film Metropolis behandelte. Für uns gab es im Projektplenum eine eigene kleinere Veranstaltung zu diesem Thema, in dem in Kurzreferaten zu ähnlichen Themen diskutiert wurde.
Eine weitere empfohlene Veranstaltung war **Gestaltung zwischen Stoff und Zeichen Der Computer als semiotische Maschine** von Frieder Nake. Hier ging es um den Computer als Maschine zur Bearbeitung von Zeichenprozessen, also in die Richtung der Semiotik. In dieser Veranstaltung machten aber nur wenige, und die auch nicht im Seminarteil, mit.

Im letzten Semester gab es keine richtige begleitende Veranstaltung, so dass wir uns voll auf unsere Proben konzentrieren konnten. Trotzdem haben wir durch die Proben und die Zusammenarbeit mit den Kulturwissenschaftlern sehr viel (auch organisatorische Dinge) gelernt, so dass man den Probenplan schon als eigene Veranstaltung in diesem Semester bezeichnen könnte.

1.3 kultureller Teil

Neben den universitären Veranstaltungen gab es auch noch kulturelle, die wir besucht haben, die zum Thema passten und uns nicht unbedingt inspiriert, aber sicher immer beeindruckt haben.
So besuchten wir im neuen Museum Weserburg die Installation *Die Sonne*, die zwar schon alt ist, aber zeigt, wie auch immer noch heute neue Medien in der Kunst eingesetzt werden. Außerdem besuchten wir eine Performance des Künstlers Stelarc in Hamburg. Dort führte er sein Exoskelett vor, also eine pneumatische Maschine, mit der er sich bewegen konnte und die ihm einem dritten Arm zur Verfügung stellte. Anschließend konnten wir noch mit ihm sprechen und ihm von unserem Projekt erzählen, wie auch er von seinen früheren Performances redete: Beispielsweise konnten einmal Menschen über das Internet seine Muskeln beeinflussen und ihn damit bewegen. Ein interessanter Mensch. Es bot sich auch die Möglichkeit, in Osnabrück eine Aufführung von Chico MacMurtrie und seinen Robotern zu besuchen. Sie war sehr beeindruckend, und es ist verständlich, wie Willi auf diese Projektidee kam. Leider waren es nur wenige Leute aus dem Projekt, die dort waren.

Ebenso zum kulturellen Teil gehört der Abschnitt zu den Projektfahrten, die wir jedes Semester unternommen haben.
Im ersten Semester ging es in die Jugendherberge nach Cuxhaven. Diese Fahrt diente vor allem dazu, uns erst einmal kennenzulernen, aber auch dazu, unsere Projektidee zu konkretisieren.
Im zweiten Semester ging es zum Camping nach Helgoland. Inhaltlich mussten wir dort unsere Projektidee überarbeiten, ansonsten haben wir neue Philosophien (den DerDuDie-ismus) und Feuerrituale (Götteropferung) erfunden.
Im dritten Semester hat Willi uns zu sich nach Berlin eingeladen. So konnten wir bewundern, wie Professoren leben. Außerdem haben wir Kleinkunst erlebt und wissen dadurch, wann Denis Geburtstag hat.
Die Projektfahrt im letzten Semester war die Fahrt zur Abschlussaufführung im Rahmen eines Stuttgarter Theaterfestivals *Schöne Aussicht*. Abgesehen vom Bühnenauf- und -abbau sowie der Proben und der Aufführung, haben wir den Abschluss gefeiert, gelernt, wie man sich im Hotel eincheckt und wann man sich Giovanni zu nennen hat.
Die einmal als allerletzte Fahrt wiederum nach Berlin fand leider nicht mehr statt, aber es wird dennoch eine kleine Abschiedsfeier geben, alleine schon, um die Projektkasse zu leeren.

1.4 Das Ergebnis

Nachdem nun also der Weg (sowie einige Begebenheiten neben diesem) beschrieben wurde, folgt nun eine kurze Beschreibung des Zieles, also des Erreichten.
Wir haben es geschafft, ein Theaterstück auf eine Bühne zu bringen. Es dauerte zwar nur ca. 15

bis 20 Minuten, aber immerhin.
Das Szenario wurde von Jörg Richard entworfen und der Hauptakteur auf der Bühne war Marc, der Cellist, aber alles andere von der Bühnenkonstruktion über Licht- und Soundtechnik bis zu allen agierenden Maschinerien stammt von der Idee bis zur Verwirklichung aus dem Projekt.
Alle zusammen, also Menschen, Maschinen und (Bühnen-)Technik agieren gemeinsam auf der Bühne, um das Zusammenspiel von Mensch-Maschine-Natur darszustellen, auch wenn die Gesamtbotschaft eher im Dunkeln bleibt. Das Stück ist einzigartig und wird (gerade wegen unserer Maschinen) nie wieder so aufgeführt werden.
Folgende Maschinen traten dabei auf:

- Das FliWaTüt
- Die Marionette im Kasten
- Die medial-reale Marionette
- Die virtuelle Marionette
- Der Mülleimer
- Elektrische Töne
- Ein Handy
- Die Mailbox
- Der Drucker

Da der Drucker ein ganz normaler Drucker war, das Handy nur eine Pappatrappe und die Mailbox nur ein Feature der elektrischen Töne, die allesamt von der Informatikseite aus betrachtet uninteressaant sind, werden in der nächsten Kapiteln nur die anderen von uns erdachten und gebauten Maschinen, alle Ergebnisse einzelner Unterprojekte, genauer beschrieben. Ebenso noch unsere Licht- und Gesamtsteuerung.
Dort werden alle technischen Feinheiten, die man größtenteils im Stück gar nicht erkennt, genauer betrachtet. An dieser Stelle muss genügen, dass diese Geräte zusammen mit dem Cellisten und der Figur zusammen spielen konnten und so das Stück von Jörg Richard umgesetzt haben.

Kapitel 2

Mülleimer

2.1 Einleitung

Durch die ersten Ideen inspiriert, eine Art surealen Raum der Maschinen zu erschaffen, durch den die Besucher hindurchgehen können, war der erste Gedanke eine Bierkiste zu entwerfen und bauen. Diese sollte mittels Sensoren und Rädern durch den Raum navigieren können. Im späteren Verlauf des Projektes bekamen wir allerdings die Erkenntnis, dass die Realisierung dieser Bierkiste unsere Möglichkeiten bei weitem überstieg, so dass wir gezwungen waren unsere Idee neu zu überarbeiten. Dies führte dann über die sogenannte ,,Puppe" zum Mülleimer. Dieser wurde im späteren Verlauf dann in die Vorführung integriert. Die einzelnen Arbeitsschritte die zur Realisierung des Teilprojekts ,,Mülleimer" führten sollen hier erläutert werden.

2.2 Bierkiste

Die ersten Spezifikationen der sogenannten Bierkiste für die Installation des Raumes, sahen ein Fahrwerk, Sensoren und eine Bierausgabe vor. Mit Motoren die ein Fahrwerk mit Rädern antreiben sollten, sollte es möglich sein die Bierkiste in jede beliebige Richtung zu bewegen. Dabei war noch keine präzise Geschwindigkeit vorgesehen. Mittels Sensoren sollte eine Orientierung im Raum geschaffen werden. Diese sollte Hindernisse erfassen und ein Ansteuern oder Ausweichen der Bierkiste ermöglichen. Wir hatten dafür Abstandsensorik geplant. Als letztes Attribut war eine geldabhängige Ausgabe von Bierflaschen vorgesehen. Diese Flaschen sollten mit einem zu diesem Zeitpunkt nicht näher erörterten Mechanismus gesichert sein und bei Münzeinwurf diese freigeben. Der gesamte Roboter war als ein sich autonom steuernder konzipiert. Es sollten also keinerlei Kabelverbindungen zum Computer vorhanden sein. Hierbei waren die Möglichkeiten einer Installation der benötigten Software zur Steuerung auf der Maschine oder eine Steuerung per Funk in Betracht gezogen worden. In jedem Fall musste eine zusätzliche Energiequelle auf dem Gefährt Platz finden.
Die erste Realisierung der Maschine erwies sich größer als angenommen. Um eine den Anforderungen gewachsene Hardware auf dem Roboter zu installieren, war eine Anzahl von sechs miteinander verschraubter Bierkästen notwendig. Hierbei hatten wir eine Aufteilung der Hardware in drei Ebenen, bestehend aus jeweils zwei Bierkästen ins Auge gefasst. Die unterste Ebene sollte die Motorik und die Energieversorgung beherbergen. In der mittleren Ebene war die Sensorik geplant und in der obersten Ebene sollte der Mechanismus für die Ausgabe der Bierflaschen mitsamt der Bierflaschen untergebracht werden.
Im weiteren Verlauf des Projekts erwies sich das Gewicht der ,,Bierkiste" und aller Hardwarekomponenten als zu hoch. Keiner der von uns in Betracht kommenden Schrittmotoren konnte das Gewicht des Roboters bewegen. Es wurde seitens der Untergruppe sehr viel Energie und Zeit darauf verwendet eine Lösung für das Antriebsproblem zu finden, bis klar wurde, dass eine Realisierung nicht im finanziellen Rahmen lag.

2.3 Luftpuppe

Wir verwendeten danach einige Zeit darauf neue Ideen und Konzepte zu sammeln, wobei die Idee eines autonomen Roboters immer im Vordergrund stand. Da unsere Mittel begrenzt waren mussten wir eine kleinere und leichtere Alternative zur vorhergehenden Idee wählen. Die Maschine sollte nur aus einer Grundplatte auf der die einzelnen Hardwarekomponenten wie Antrieb, Steuerung und Sensoren angebracht waren und einer Mechanik bestehen, die einen Sinn erfüllen sollte um einen Beitrag zum Gesamtkonzept des Projekts herzustellen. Angeregt durch die Idee unseres Proffesors W. Bruns fingen wir an mit einer sogenannte Luftpuppe zu experimentieren. Diese Puppe sollte eine aus Stoff bestehen Gestalt sein, die Gliedmaßen und einen Kopf besitzt. Durch ein Gebläse sollte diese sich aufrichten. Eine solche ,,Luftpuppe" war als etwa 5-10 m großes Modell auf dem Freimarkt zu beobachten. Um ein ausreichendes Gebläse herzustellen, bauten wir aus alten Computern die Lüfter aus. Zum Zweck des Ausprobierens wurden die Lüfter auf eine angefertigte Grundplatte geschraubt. Die ersten Versuche starteten wir mit Plastiktüten, die wir zu solchen sogenannten Puppen verklebten. Als eine Art Nebeneffekt entstand bei diesen Versuchen eine Art Hovercraft, also eine sich durch Luftkissen fortbewegendes Transportmittel. Leider ließ sich diese Antriebsform sehr schlecht weiterentwickeln, so dass die Entwicklung in dieser Richtung eingestellt wurde.
Nachdem alle Versuche eine solche ,,Luftpuppe" zu konzipieren erfolgreich waren, begannen wir mit der Auswahl des Materials und der genauen Betrachtung des Gefährts auf dem die Luftpuppe installiert werden sollte.

2.4 Mülleimer

Durch die Kooperation mit Kulturwissenschaftlern und der genauen Ausformung unserer Vorführung wurde dann die Idee geboren die sogenannte Luftpuppe aus einer Art Eimer kommen zu lassen. Diese Idee entwickelten wir weiter, bis wir auf ein Gefährt kamen, dass vom Aussehen her an einen Mülleimer erinnerte. Der grundlegende Gedanke bestand darin, dass man Müll in das Gefährt tun konnte, der dann beim Aufblasen der Luftpuppe herausgeschleudert wird, die sich nun vollständig aufrichtet.
Zu diesem Zeitpunkt wurde auch der Entschluss gefasst den Roboter mit einer Kabelverbindung über einen Computer zu steuern, da eine Autonomie der Maschine nicht mehr praktikabel erschien. So konzipierten wir ein Gefährt, das von Schrittmotoren angetrieben wurde und ein Gebläse enthielt um eine Luftpuppe aufblasen zu können, die dann aus dem Gefährt heraus kam. Diese Maschine sollte mittels einer Kabelverbindung zum Computer über eine Software gesteuert werden.
Siehe auch Abbildung 2.1

2.5 Aufbau des Mülleimers

Der Mülleimer besteht aus einer Grundplatte aus Aluminium auf der acht Ventilatoren angebracht sind. Unter dieser Platte befinden sich die gegenüber liegenden Schrittmotoren, die mittels eine Getriebeübersetzung zwei Räder antreiben. Zur Stabilisierung befinden sich noch zwei Stützräder unter der Grundplatte. Von dieser Basis ausgehend sind an jeder Ecke vier Rundholzstangen von etwa einem Meter Länge vertikal installiert. Um eine Beschädigung der Ventilatoren oder der Stoffpuppe zu verhindern montierten wir ein Gitternetz auf halber Höhe an den Rundholzstangen. Oben auf den vier Holzstangen wurde eine zweite Platte angebracht die eine große runde Öffnung enthält. An dieser Öffnung haben wir von innen die Luftpuppe befestigt. Als Verkleidung wurde ein starker Stoff um das Gefährt gewickelt, der mittels eines Klettstreifens an der Ober- und Unterkante befestigt ist und durch einen vertikal eingenähten Reißverschluss geschlossen wird. Als Zusatz haben wir eine Innenbeleuchtung angebracht, so dass die Puppe im Dunkeln leuchtet. An der Vorderseite der Maschine befindet sich vertikal

angebracht eine weitere Aluminiumplatte, an der die gesamte Steuerung des Mülleimers befestigt ist. Dafür haben wir sogenannte Blackboxen benutzt, in denen die Steuerplatinen untergebracht sind. Diese Blackboxen sind mit Stromanschlussbuchsen und Hardwareschnittstellen für die Steuerlogik ausgestattet.
Das Gefährt musste mit drei Spannungsquellen versorgt werden. Für die Steuerungslogik benötigten wir eine 12V Spannung, ebenfalls war eine 24V Stromquelle erforderlich, um die Ventilatoren zu betreiben und für die Lichtquelle brauchten wir 220V. An der Unterseite der Grundplatte hatten wir daher Anschlüsse zur Stromversorgung angebracht aus denen alle Komponenten versorgt wurden. Zur externen Stromversorgung haben wir zwei stufenlos regulierbare Netzteile und eine herkömmliche Steckdose benutzt. Ebenfalls hatten wir eine sogenannten RS232 Hardwareschnittstelle angebracht um die geplante Kommunikation zwischen Mülleimer und Computer gewährleisten zu können.
Für die Steuerung der Motoren, der Ventilatoren und der Lampe haben wir einen Mikrocontroller genommen. Mit diesem Controller waren wir in der Lage über die Schrittmotorsteuerung die Motoren in Richtung und Geschwindigkeit zu steuern. Bei den Ventilatoren und der Lampe benötigten wir lediglich einen An/Aus-Zustand der mittels einer Relaissteuerung über dem Mikrocontroller gesteuert wurde. Der Controller wurde über eine spezielle Software vom Computer aus mit Assembler programmiert und arbeitete dann autonom und ohne jegliche Computerverbindung das gespeicherte Programm ab. Da allerdings unser Programm über eine Schnittstelle zum Computer verfügen musste, um in Echtzeit agieren zu können, mussten wir eine Kabelverbindung etabliert halten. Die Planung sah vor die Maschine über eine Oberfläche vom Computer aus zu steuern. Diese Software konnte aber aus Zeitmangel nicht mehr komplett fertig gestellt werden. So waren wir leider gezwungen eine Alternative Steuerung zu erstellen. Dies gelang uns mit Hilfe eines Joysticks, den wir mit einigen Modifikationen an den Roboter angeschlossen haben, um diesen zu steuern. Damit war es möglich das Gefährt zu bewegen, die Ventilatoren an und aus zu schalten und die Lampe zu bedienen.

2.6 Mikrocontroller

Zur Steuerung des gesamten Gefährts mit allen Hardwareelementen benutzten wir einen 8-Bit-Mikrocontroller, den AVR90S8515 von Atmel. Dieser besitzt ein programmierbares Flash-ROM, in welches Maschinencodeprogramme übertragen werden können. Als Startvorlage verwendeten wir ein Evaluierungskit(STK200) von Atmel. Kernstück dieses Kits ist ein Evaluierungsboard mit besagten Mikrocontroller.

Zur Programmierung des Controllers benutzten wir verschiedene mitgelieferte Programme. Diese lieferten eine Entwicklungsumgebung zur Programmierung des Controllers in Assembler und ein Werkzeug zum übertragen des erzeugten Maschinencodeprogramms in das Flash-ROM des Controllers.

Um eine Kommunikation zum Computer herzustellen, besitzt der Controller die Möglichkeit eine Verbindung über Kabel an die serielle Schnittstelle des Computers herzustellen. Als Ausgabe besitzt der Controller fünf sogenannte Ports, die jeweils acht ansteuerbare Pins besitzen. Diese können mit Hilfe einer 8 Bit Kodierung angesprochen werden. Dabei ist das setzen des jeweiligen Bits gleichzusetzen mit dem schalten einer Spannung an dem jeweiligen Pin. Das bedeutet durch das senden eines acht Bit Wertes an einen der sogenannten Ports war es möglich die Steuerung zu realisieren. Um die Steuerung der Motoren zu bewerkstelligen, war eine vier Bit Kodierung ausreichend. Dabei gingen wir nach folgendem Schema vor :

Bit 0 : Masse. Musste immer geschaltet sein.

Bit 1 : Motor ein/aus. 0 für ein, 1 für aus.

Bit 2 : Motordrehung rechts/links. 0 für Rechts-, 1 für Linksdrehung

Bit 3 : Motor langsam/schnell. 0 für schnell, 1 für langsam.

Bit 4 : Nicht verwendet. Standardwert 0.

Bit 5 : Nicht verwendet. Standardwert 0.

Bit 6 : Nicht verwendet. Standardwert 0.

Bit 7 : Nicht verwendet. Standardwert 0.

Für die Steuerung der beiden Motoren verwendeten wir zwei Ports.
Des weiteren verwendeten wir einen weiteren Port für die Steuerung der anderen Hardwarekomponenten Licht und Ventilatoren. Für das Licht benötigten wir ein Bit, und für die Ventilatoren zwei, da diese in zwei Gruppen angeordnet waren. Bei einem weiteren Bit für die Masseleitung, war eine vier Bit Kodierung erforderlich. Dabei war folgendes Schema maßgebend :

Bit 0 : Masse. Musste immer geschaltet sein.

Bit 1 : Ventilatorgruppe A ein/aus. 1 für aus, 0 für ein.

Bit 2 : Ventilatorgruppe B ein/aus. 1 für aus, 0 für ein.

Bit 4 : Licht ein/aus. 1 für aus, 0 für ein.

Bit 5 : Nicht verwendet. Standardwert 0.

Bit 6 : Nicht verwendet. Standardwert 0.

Bit 7 : Nicht verwendet. Standardwert 0.

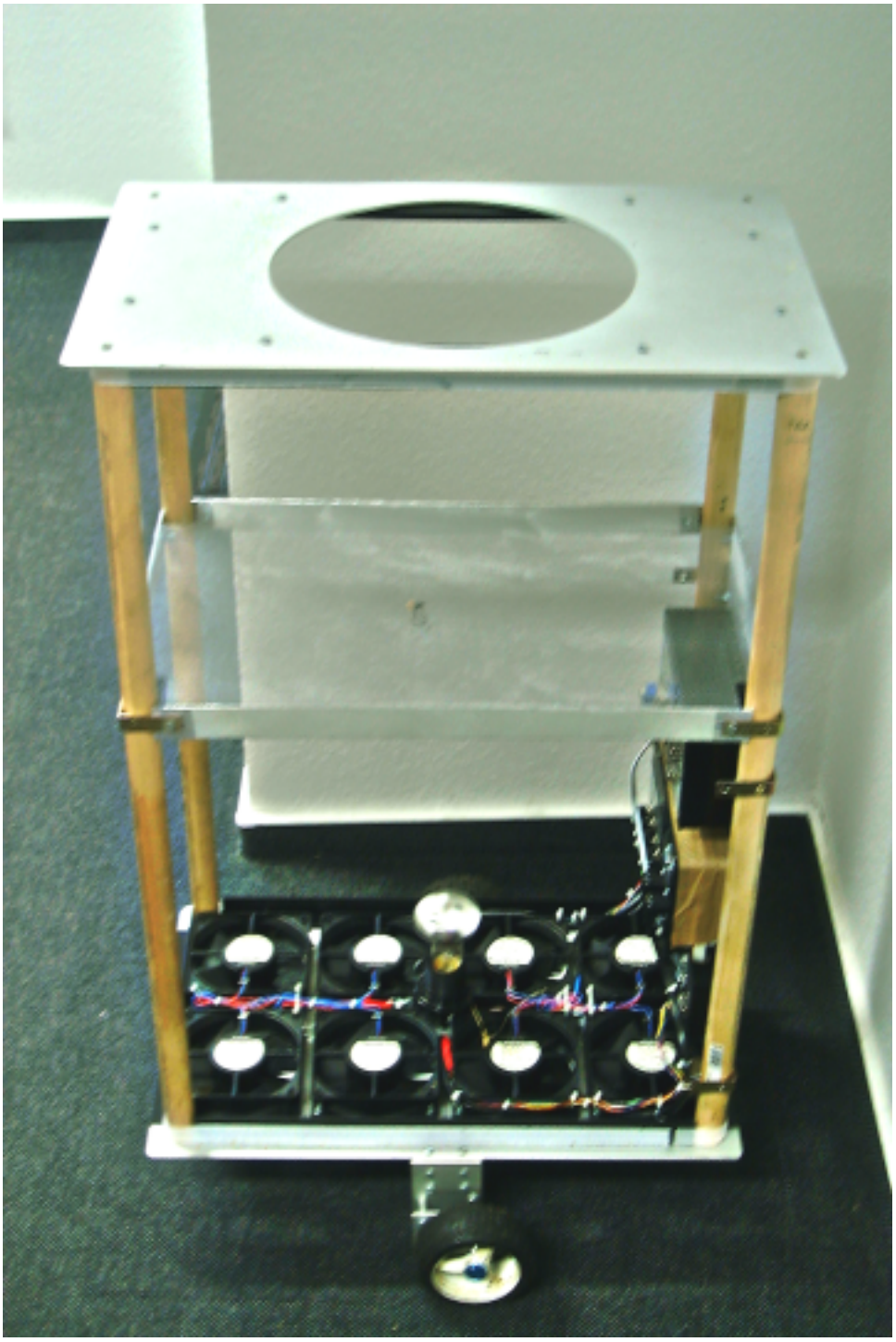

Abbildung 2.1: Der Mülleimer ohne Verkleidung

Die Platine des Mikrocontrollers war in einer Blackbox untergebracht und hatte folgende Anschlüsse :

- 12 Volt Anschluss
  Für die Stromversorgung der Platine.
- RS232 Schnittstelle
  Zur Kommunikation mit dem Computer.
- Relaissteuerung
  Zur Steuerung der Relais, die zur Bedienung des Lichtes und der Ventilatoren führte. Hierfür verwendeten wir einen 9-poligen Stecker.
- Motorsteuerung rechts
  Zur Ansteuerung des rechten Motors. Hierfür verwendeten wir eine 9-polige Buchse.
- Motorsteuerung links
  Zur Ansteuerung des linken Motors. Hierfür verwendeten wir eine 9-polige Buchse.

Siehe Abbildung 2.2 und 2.3

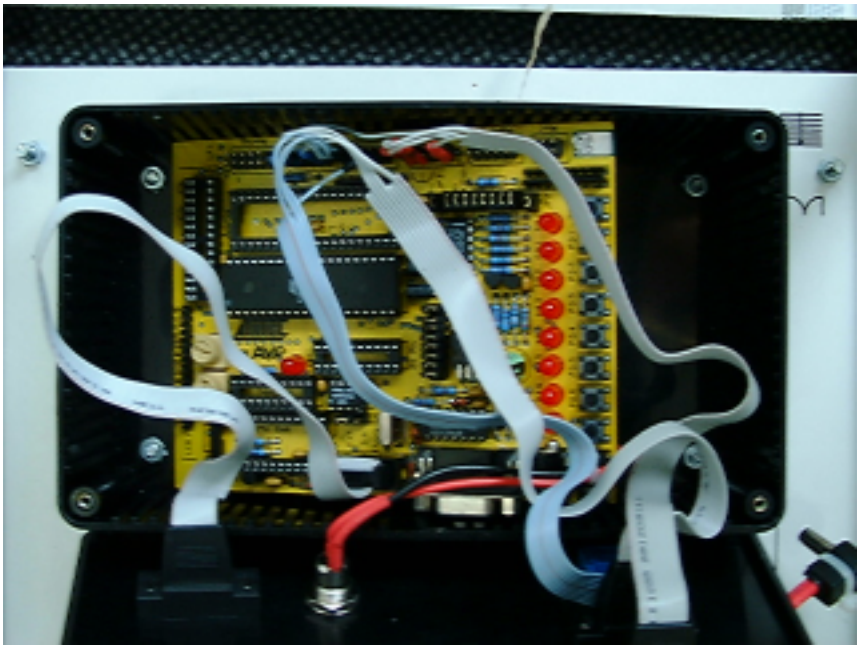

Abbildung 2.2: Der Mikrocontroller

2.7 Schrittmotoren

Als Schrittmotoren haben wir vier Phasen Schrittmotoren genommen die wir mittels einer Schrittmotorsteuerung ansteuern konnten. Diese Steuerung sollte die Motoren in der Art ansteuern, dass diese sich grundsätzlich mit oder gegen den Uhrzeigersinn drehen sollten und nicht mehr in Schritten bewegt werden müssen. Dabei stellte die Schrittmotorsteuerung folgende Funktionen zur Verfügung :

- An/Aus

Abbildung 2.3: Die Blackbox des Mikrocontrollers

- Schnell/Langsam
- Rechts-/Linksdrehung bzw. Drehung mit oder gegen den Uhrzeigersinn

Die Steuerung der Schrittmotoren hat fünf Eingänge, zwei zur Stromversorgung und drei zur Steuerung der oben genannten Funktionen. Diese Eingänge wollten wir dann mittels des Mikrocontrollers ansteuern. An den Ausgängen der Steuerung wurden die verschiedenen Phasenleitungen des Schrittmotors angebracht.
Zuerst versuchten wir den Antrieb durch das direkte Anbringen von Reifen aus dem Modellbaubereich an der Achse der Motoren zu realisieren. Da die aufgewendete Kraft der Motoren nicht ausreichte, mussten wir allerdings eine Übersetzungsstufe mit Hilfe von Zahnrädern bauen. Dabei haben wir ein kleines Zahnrad an die Achse des Schrittmotors festgeschraubt, das ein größeres Zahnrad antrieb, an dem die Räder des Mülleimers befestigt waren.
Die Schrittmotorsteuerung war in einer Blackbox untergebracht mit den folgenden Anschlüssen :

- Logik
  9-poliger Stecker für den Anschluss an den Mikrocontroller oder die Handsteuerung.
- Motor
  9-poliger Stecker für den Anschluss des Schrittmotors und gleichzeitiger Stromversorgung.

Siehe Abbildung 2.4

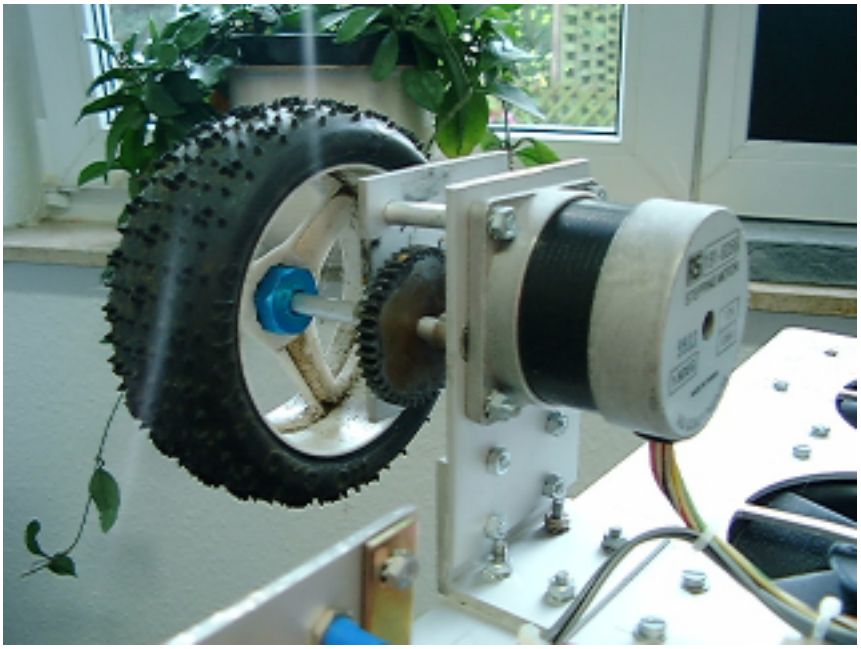

Abbildung 2.4: Schrittmotor mit Getriebe und Rad

2.8 Relaissteuerung

Um das Licht und die Ventilatoren steuern zu können benötigten wir eine Relaissteuerung, die es uns ermöglichte besagte Komponenten ein- bzw. ausschalten zu können. Dafür mussten wir eine Relaissteuerung entwickeln, die durch den Mikrocontroller angesprochen werden konnte. Da die 5V Spannungsquelle des Mikrocontrollers nicht ausreichte um die Relais sicher halten zu können integrierten wir gleichzeitig eine Verstärkerschaltung.
Damit war es uns ohne weiteres möglich über den Controller bzw. über die Handsteuerung die Hardware Licht und Ventilatoren anzusteuern.

Funktionsweise:

- Diode Dl
  Verhindert das bei einem Verpolen oder einem Kurzschluss möglicherweise eine Spannung am Eingang anliegt, die so evtl. die Mikrocontrollersteuerung schädigen könnte.

- Widerstände R1,R2
  Die Wiederstände R1 (15kΩ) und R2 (4,7kΩ) teilen die Eingangsspannung so auf, dass an der Basis des Transistors T1 genau 0V oder 0,7V (Ein / Aus) anliegen.

- Transistor T1
  Der Transistor T1 schaltet mit Hilfe der 5V Eingangsspannung die spannungsfesten 12V Verstärkerspannung, damit ist das sichere Anziehen des Relais gewährleistet.

- Relais K1
  Das Relais K1 dient zum schalten der schaltbaren Leitung.

- Generell
  Die Eingangs-5V werden durch die beiden Wiederstände R1 + R2 so aufgeteilt, dass an der Basis des Transistors T1 genau 0,7V anliegen. Bei anliegen dieser Spannung schaltet der Transistor T1 die 12V Verstärkerspannung durch, somit zieht das Relais K1 an. Fallen die 5V Eingangsspannung ab, so schließt der Transistor T1 und somit fällt auch das Relais

ab.

Die Relaissteuerung war in einer Blackbox mit den folgenden Anschlüssen untergebracht :

- 240 Volt Eingang
  Zur Durchschaltung für die Lichtquelle.
- 240 Volt Ausgang
  Als Anschluss für die Lichtquelle.
- 12 Volt Eingang
  Zur Stromversorgung der Relais.
- 24 Volt Eingang
  Zur Durchschaltung für die Ventilatoren.
- zwei 24 Volt Ausgänge
  Zur Stromversorgung der Ventilatorgruppen A und B.

Siehe Abbildungen 2.5,2.6 und 2.7

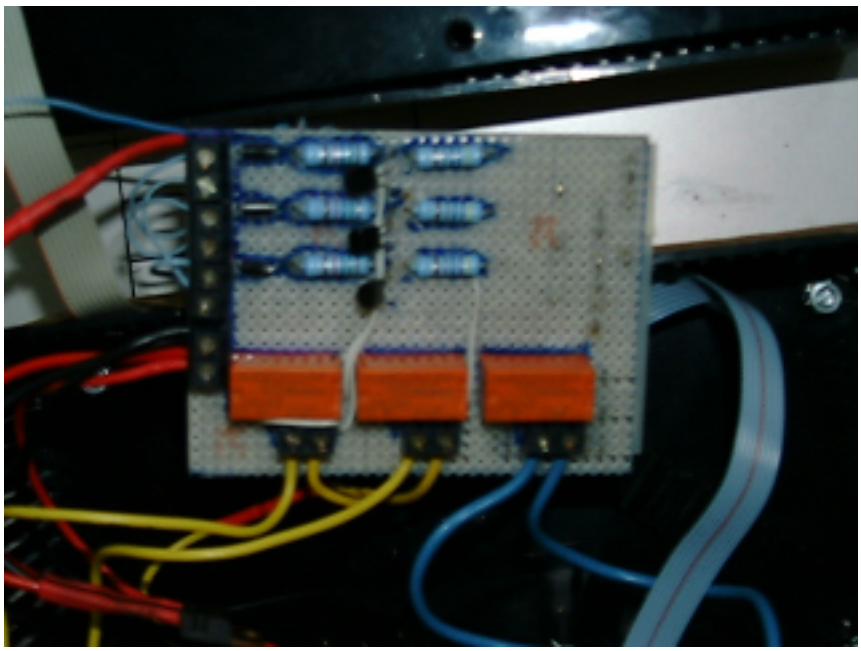

Abbildung 2.5: Platine der Relaissteuerung

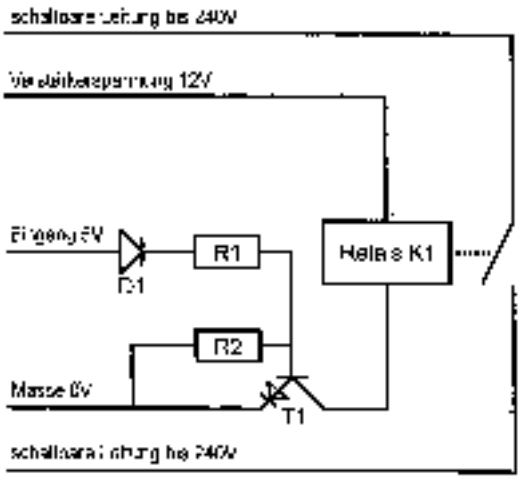

Abbildung 2.6: Schaltbild zur Realaisteuerung

Abbildung 2.7: Die Blackbox der Relaissteuerung

2.9 Joysticksteuerung

Um eine schnelle und effiziente Lösung zur Steuerung des Gefährts zu präsentieren, bauten wir eine Handsteuerung. Dazu haben wir eine sogenannten Joystick modifiziert. Hierfür haben wir die diesen einfach an die Eingänge der Relaissteuerung angeschlossen, an dem der Mikrocontroller angeschlossen ist. Dabei musste die Handsteuerung das setzten der einzelnen Bits und der damit verbundenen Pins simulieren. Die Bewegungen wurden durch einen Steuerknüppel vorgenommen. Dieser hatte acht Richtungsmöglichkeiten :

- vorne
- hinten
- rechts
- links
- Linkskurve nach vorne
- Rechtskurve nach vorne
- Linkskurve nach hinten
- Rechtskurve nach hinten

Dabei haben wir durch das Benutzen von Logikschaltungen diesen Richtungen bestimmten Verhalten der Motoren zugeordnet :

Richtung	Motorverhalten
vorne	Motor A rechtsdrehung, Motor B Linksdrehung
hinten	Motor A Linksdrehung, Motor B Rechtsdrehung
rechts	Motor A Rechtsdrehung, Motor B Rechtsdrehung
links	Motor A Linksdrehung, Motor B Linksdrehung
Linkskurve nach vorne	Motor A Stopp, Motor B Linksdrehung
Rechtskurve nach vorne	Motor A Rechtsdrehung, Motor B Stopp
Linkskurve nach hinten	Motor A Stopp, Motor B Rechtsdrehung
Rechtskurve nach hinten	Motor A linksdrehung, Motor B stop

Das Einschalten des Lichtes und die Geschwindigkeitsregulierung der Motoren, sowie das An- und Ausschalten der Ventilatoren wurden durch separate Knöpfe am Joystick realisiert. Wir haben hierfür drei Knöpfe gebraucht um die zwei Ventilatorgruppen und das Licht anzusteuern. Bei gedrücktem Knopf wurde die jeweilige Komponente eingeschaltet. Die Handsteuerung wurde an den sogenannten Lockiganschlüssen der Blackboxen Motor links bzw. rechts und der Relaissteuerung angesteckt und simulierte den Controller. So war die Maschine komplett durch den Joystick steuerbar und genügte den Ansprüchen der Theateraufführung. Siehe Abbildung 2.8

2.10 Computersteuerung

Die Planung sah vor, den Mülleimer in Echtzeit per Computer zu steuern. Aus diesem Grund begannen wir eine grafische Oberfläche für das Steuerungsprogramm - unter dem Betriebssystem Windows - zu entwickeln. Diese Oberfläche sah mehrere Funktionalitäten vor. Zum Einen sollte der Mülleimer über eine direkte Tasteneingabe gesteuert werden, zum anderen sollten

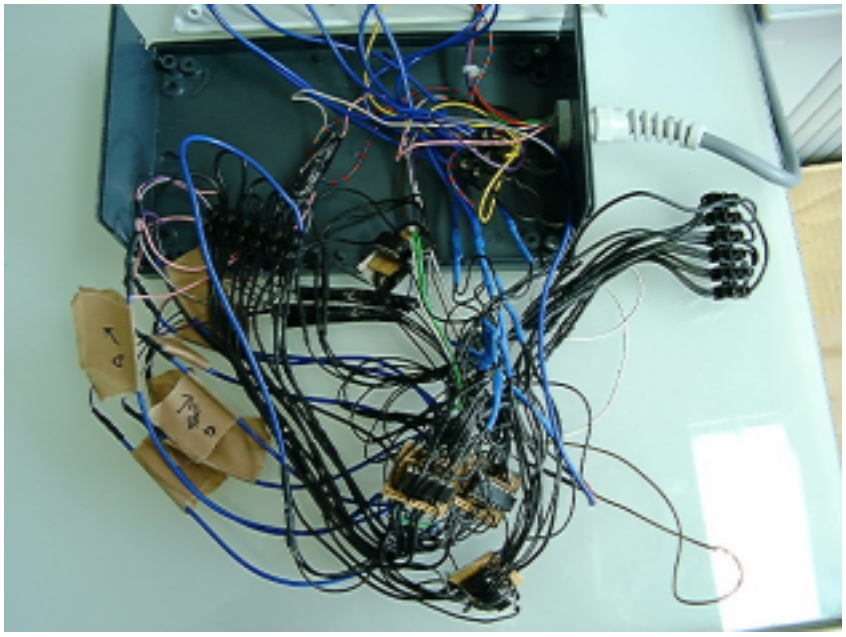

Abbildung 2.8: Die Handsteuerung

verschiedene Sequenzen ablaufbar sein. Dazu programmierten wir eine Oberfläche, die alle Eigenschaften des Roboters abbildeten. Damit war es möglich die Maschine in Echtzeit vor und zurück zu bewegen, Links- und Rechtsdrehungen zu vollziehen, das Licht und die Ventilatoren ein- und auszuschalten und zu guter Letzt konnte die Geschwindigkeit variiert werden.
Um auch vorgefertigte Sequenzen ablaufen lassen zu können, entwickelten wir noch einen Sequenzer, mit dessen Hilfe Sequenzen eingegeben werden konnten und bei Knopfdruck abliefen. Dabei fanden die gleichen Grundsteuerungselemente wie bei der Echtzeitsteuerung Verwendung. Wie sich später bei den Proben des Theaterstückes herausstellte, fand der Sequenzer keine Anwendung. Die Interaktion mit dem Schauspieler war so spontan, dass eine vorgefertigte Sequenz nicht die nötige Flexibilität erreichte.
Es war weiterhin geplant die Kommunikation zum Mülleimer über ein Kabel und die serielle Schnittstelle zu realisieren. Dies konnte wegen Zeitmangel allerdings nicht mehr realisiert werden. So wurde nur die grafische Oberfläche fertiggestellt.
Auf Seiten der Maschine sollte demnach ein Assemblerprogramm etabliert werden, dass die über das Kabel und der serielle Schnittstelle gesendeten Signale interpretieren kann und diese als Steuerung umsetzt. Auch dieser Teil der Programmierung wurde nicht umgesetzt. Siehe Abbildungen 2.9 und 2.10

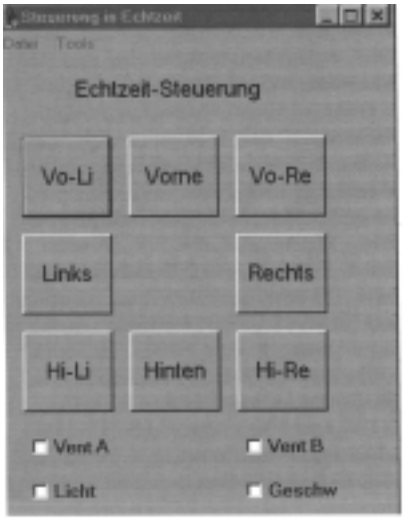

Abbildung 2.9: Grafische Oberfläche der Echtzeit-Steuerung

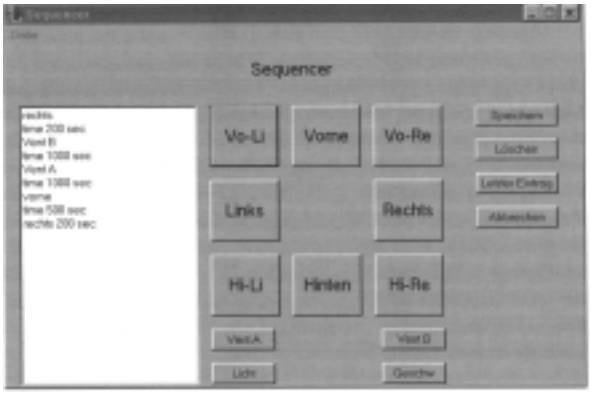

Abbildung 2.10: Grafische Oberfläche des Sequencers

Kapitel 3

Master Control Program

3.1 Entstehung

Im Verlauf des letzten Projekt-Semesters haben ein paar Mitglieder des Projektes festgestellt, daß zwar einige interessante Teilprojekte im Entstehen sind, jedoch keine zentrale Steuerung, ob nun zentraler oder dezentraler Natur, vorhanden ist.

Die Fragestellung, wie das Projekt als ganzes nun gesteuert werden soll wurde im weiteren Verlauf näher vertieft. So schienen uns verschiedene Vorgehensweisen für sinnvoll. Zum einen hätte man die komplette Steuerung dezentral halten können, so das quasi jedes Teilprojekt die Möglichkeit hat, ein anderes Teilprojekt über bereit gestellte Schnittstellen zu steuern sowie dessen Zustand abzufragen.

Ein fest vordefinierter Ablauf wäre mit dieser Methodik aber schwer zu realisieren gewesen. Die Aktionen hätte man in den einzelnen Projekten als Regeln definieren können. Dies hätte z.B. so Ausgesehen, dass der SoundRaum einen gewissen Sound spielt, wenn das linke Marionettenbein eine gewisse Position überschritten hat. Diese Regeln könnte man in allen Teilprojekten einbauen so das am Ende eine etwas undeterministischer Ablauf der Fall wäre.

Für diese Realisierung wäre aber einiges an Arbeit für die einzelnen Teilprojekte angefallen. Diese müssten die Funktionalität besitzen um ein Regelwerk zu definieren sowie die verschiedenen Teilprojekte anzusteuern. Dieser zusätzliche Zeitaufwand schien uns in Anbetracht der kurzen, noch verbleibenden Zeitspanne bis zur Präsentation des Projektes als zu aufwendig. Des weiteren sind die Projekte recht unterschiedlich ausgefallen. So kann der Soundraum z.B. mehrere Clients gleichzeitig haben. Bei der Marionette kann aber nur einer zur Zeit die Kontrolle über die Marionette haben.

Zu diesem Zeitpunkt haben wir jedoch, zusammen mit den Kulturwissenschaftlern, bereits einen festen Ablauf der Vorführung ausgearbeitet. Aus diesem Grund haben wir uns dann für ein wesentlich einfacheres, zentrales Gerüst entschieden das in dem noch zur Verfügung stehenden Zeitraum realisierbar war.

3.2 Prinzip des MCP

Bei dem von uns gewählten Ansatz stellt jedes Projekt, welches über das MCP (Master Control Program - so der Name der zentralen Steuerungsinstanz) angesprochen wird, Schnittstellen zur Verfügung über die das Projekt gesteuert werden kann. Dabei sollen diese Schnittstellen über Mechanismen verfügen, die über die Wahl des eigentlichen Kommunikationskanals abstrahieren.

So erfolgt die Kommunikation mit dem SoundServer z.B. über CORBA. Für den Lichtserver wird das TCP/IP Protokoll verwendet. Die eigentliche Kommunikation geschieht innerhalb der bereitgestellten Klassen (Schnittstellen), so das für das MCP eine gewisse Transparenz der Funktionsaufrufe gegeben ist.

Dies wird in der Abbildung 3.1 noch mal veranschaulicht. Die bereitgestellten Schnittstellen werden zum Master Control Programm hinzugebunden, was durch die Kästen innerhalb des MCP verdeutlicht wurde.

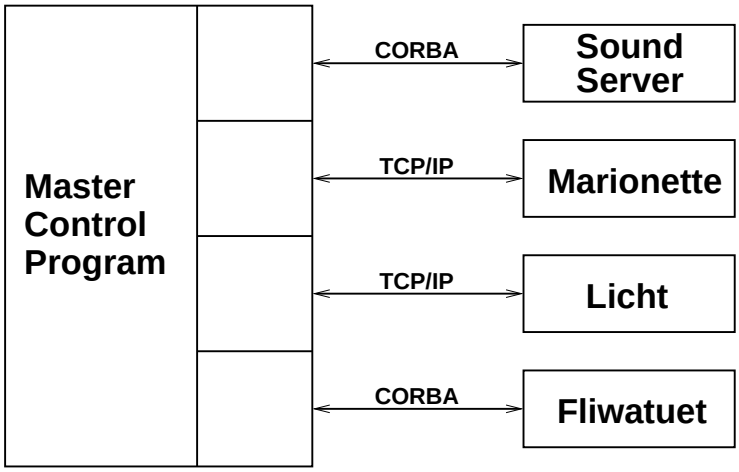

Abbildung 3.1: Struktur des Master Control Programmes

Für die Steuerung war anfangs noch ein Event System vorgesehen. Die Idee dahinter war, dass man bestimmte Regeln definieren kann. Dies sähe in der Praxis so aus, dass verschiedene Events der Teilprojekte mit bestimmten Aktionen verknüpft sind, wie bereits weiter oben angerissen. Diese Funktionalität ist aber für die eigentliche Aufführung völlig überflüssig so das wir diesen Aspekt wieder außen vor gelassen haben.

Eine weitere Möglichkeit, die jedoch letztendlich nicht realisiert wurde, ist die Option den kompletten Ablauf vorzuprogrammieren, so das man bei der Aufführung dann einen recht entspannten Job hat und lediglich aufpassen muss, dass keine Cola in den Rechner geschüttet wird oder die Netzwerkkabel rausgerupft werden. Dies wäre relativ schnell und einfach zu realisieren gewesen. Jedoch war der Ablauf der Vorführung keineswegs fest. Das Timing der Einsätze musste relativ genau kommen. Diese Einsätze waren jedoch keineswegs an fest definierte Zeitpunkte gebunden, sondern sollten vielmehr dann geschehen wenn bestimmte Zustände eintraten (z.B. wenn der Cellist eine bestimmte Aktion macht oder ein besonderer Ton gespielt wird).

Im MCP definiert man sogenannte Aktionen. Diese Aktionen werden durch beliebige symbolische Namen identifiziert. Für diese Aktionen kann man dann eine Befehlskette definieren die ausgeführt werden sollen. Abbildung 3.2 zeigt ein Screenshot des MCP bei der Arbeit.

In der linken Spalte sind dabei die bereits definierten Aktionen zu sehen. Wenn man einen dieser Einträge anklickt so erscheint in der rechten Spalte die Liste der mit diesem Namen verbundenen Befehle. Die Befehle in der rechten Spalte kann man durch Doppelklick einzeln auslösen. Durch Doppelklick auf einen Entrag in der rechten Spalte wird die komplette Befehlskette ausgeführt. Die Reihenfolge der Einträge kann man mittels der Buttons, welche durch Pfeile gekennzeichnet sind, ändern.

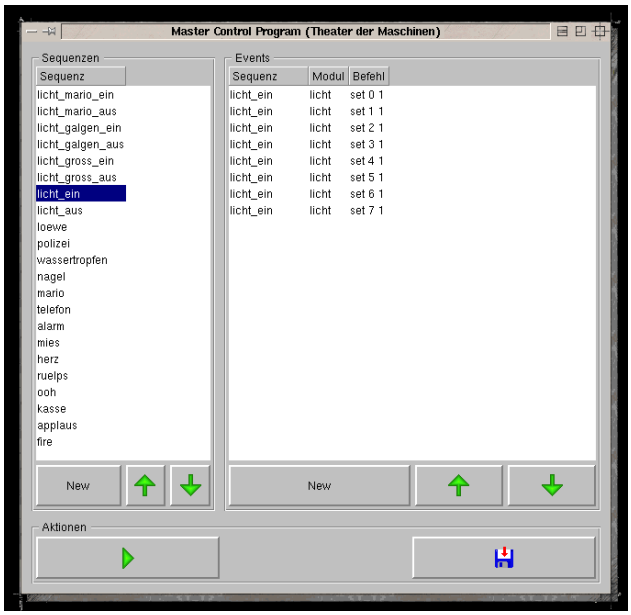

Abbildung 3.2: Screenshot vom MCP bei der Arbeit

Durch Rechtsklick auf einen der Einträge kann man diesen Editieren oder Löschen. Die Einträge welche in der rechten Tabelle aufgelistet sind bestehen aus drei Teilen. Da wäre zum einen der frei wählbare Name des gesamten Eintrages. Ein weiterer Eintrag bezeichnet das genaue Modul, an welches der Befehl gesandt werden soll. Die folgende Tabelle zeigt die, dem MCP zur Verfügung stehenden Module.

Modul	Bedeutung
sound	Sound Modul
licht	Licht Modul
mario	Marionette
mcp	eingebaute MCP Routinen

Ein weiterer Eintrag spezifiziert den genauen Befehl, der an das angegebene Modul gesandt werden soll. Die Bedeutung dieses Eintrages ist abhängig vom verwendeten Modul.

Bei dem Lichtmodul ist dieser Eintrag streng an das Kommunikationsprotokoll vom Lichtserver gehalten. Ein Befehl, gerichtet an das Licht Modul, hat die Form: `set <licht nr.> <status>`. Die Licht Nr. ist eine Zahl von 0 bis 7 mit denen die verschiedenen Relais geschaltet werden. Status gibt dabei den neuen Schaltzustand an. Eine 0 steht für Licht aus und eine 1 für Licht ein.

An das Soundmodul kann man zwei verschiedene Befehle senden. Mit `'play <soundname>'` kann das Abspielen eines Sounds veranlasst werden. Durch `'stop <soundname>'` wird der gewünschte Sound entsprechend angehalten. Die Namen müssen natürlich dem Soundserver bekannt sein, d.h. in seiner Skript Datei definiert sein. Man könnte hier noch weitere Befehle für das Sound-Modul einbauen, wobei diese die Funktionalität des ganzen nicht sonderlich erweitert, da der Soundserver ja bereits eine komplette Skriptsprache bietet mit der man bestimmte Aktionen zu bestimmten Zeitpunkten des Abspielens auslösen kann.

Bei der Marionette ist dies analog zum Soundraum. Man kann mit `'play <sequenz>'` eine vordefinierte Sequenz abspielen und mit `'stop'` diese wieder stoppen. Auch hier müssen die Sequenzen vorher im Marionettenmodul definiert sein.

Das MCP bietet ausserdem noch ein eigenes Modul an, mit dem bestimmte Funktionen aus-

gelöst werden können. Das MCP Modul unterstützt zur Zeit nur den Befehl `'sleep <seconds>'` mit der die Abarbeitung der Befehlskette für die gewünschte Zeit angehalten wird.

Die erstellten Aktionen werden in einer Textdatei abgespeichert. Diese Textdatei muss man in der Regel nicht manuell editieren, da diese mit Hilfe des MCP verändert werden kann. Der Form halber wird in Abbildung 3.3 die formale Grammatik noch mal abgebildet.

```
START       = INSTRUCTION START | ''

INSTRUCTION = IDENT ':' MODUL

LETTER      = 'a'-'z' | 'A' - 'Z'
SPECIAL     = '_' | '-'
DIGIT       = '0'-'9'
IDENT       = LETTER (LETTER | DIGIT | SPECIAL)*

MODUL       = MCP | SOUND | LICHT | MARIO

MCP         = 'mcp:sleep' DIGIT
SOUND       = 'sound:play ' IDENT | 'sound:stop ' IDENT
LICHT       = 'licht:set ' DIGIT ' ' DIGIT
MARIO       = 'mario:play ' IDENT | 'mario:stop'
```

Abbildung 3.3: Die Grammatik für den MCP Ablauf

3.3 Fazit

Bei der Realisierung des MCP wurde extrem Wert auf Einfachheit und Realisierbarkeit geachtet. Dies lag zum grössten Teil an dem knappen, noch verbliebenen Zeitraum bis zur Projektpräsentation und zum anderen waren wir alle noch mit unseren eigenen Projekten stark beschäftigt.

Bei der eigentlichen Aufführung wurde das MCP lediglich zum Steuern des Soundraumes benutzt. Der Grund, warum das MCP am Ende nicht zur zentralen Steuerung des gesamten Projektes zum tragen kam, lag zum einen daran das nicht alle Projekte eine Schnittstelle zur externen Steuerung zur Verfügung stellten. So hatte der Mülleimer keinerlei Möglichkeiten zur computergestützten Steuerung vorgesehen. Die CORBA Schnittstelle zum Fliwatuet wurde aufgrund der Kürze der Zeit nicht mehr fertig.

Aber abgesehen von dem Fehlen der Schnittstellen zu jenen Projekten wäre es auch utopisch den gesamten Ablauf zentral vom MCP aus zu steuern. Bei der Aufführung gab es keine zeitlich genau definierten Einsätze so das man fast jeden Einsatz per Hand starten musste. Allein die Steuerung des Soundraumes zu den gegebenen Einsätzen war ein Vollzeitjob. Mit der Steuerung aller Komponenten zum richtigen Zeitpunkt wäre man schlichtweg überfordert gewesen.

Kapitel 4

Der SoundRaum

4.1 Ideenfindung

Die Idee des 3-Dimensionalen Soundraumes entstand wie die meisten der Teilprojekte auf der ersten Projektfahrt in Duhnen. Hier haben alle Projektmitglieder über das selbstzudefinierende Ziel des Projektes Theater der Maschinen nachgedacht. Nach ein paar produktiven Sitzungen die regelmäßig stattfanden, war die Idee geboren, eine Aufteilung des relativ großen Projektes in mehrere Teilprojekte vorzunehmen. Dies kam vor allen dadurch zustande, das die verschiedenen Projektteilnehmer sich alle ihre eigenen Gedanken zum Projektziel gemacht haben. Ergebnis war eine mehr oder weniger zusammengewürfelte Ansammlung von Ideen. Diese wurden im großen Rahmen von allen diskutiert und schon im Vorfeld nach der Durchführbarkeit gefiltert. Allzu abwegige Vorschläge wurden daher relativ schnell fallen gelassen, meist waren sie auch nicht sehr ernst gemeint, oder aber erforderten einen sehr großen Zeit- und Personalaufwand und waren deshalb in den zur Verfügung stehenden 2 Jahren nicht umzusetzen.

Die Idee, die uns auf dieser Projektfahrt gekommen ist, und die wir in vielen Diskussionen und Unterhaltungen entwickelt haben, soll im Folgenden dargestellt werden. Dazu sind einige allgemeine Überlegungen hier aufgeführt, die den Weg unserer Gedanken bis zur "fertigen" Vorstellung kennzeichnen. Es soll zunächst eine kleine Einführung in die Welt des Schalls und der aktuellen Abstrahlungstechniken vermittelt werden. Dies dient zum Verständnis der Idee des dreidimensionalen Soundraumes.

4.2 Verschiedene Abstrahlungstechniken für Musik

4.2.1 Mono

Mono-Sound ist bekanntermaßen eine punktuelle Soundquelle, wie sie heute allerdings nur noch recht selten in Musikanlagen oder sonstigen sich mit Musik beschäftigenden Konzepten finden. Meist liegt die Anwendung in einfachen Informationssystemen, die keine gesteigerte Qualität benötigen. Hier lässt sich als Beispiel sprechende Infoboxen oder sonstige sprachübermittelnde Systeme nennen, die Wert auf die zu übertragende Information und nicht auf die räumliche oder sonst wie geartete Qualität legen. Der Aufbau, den man sich hierzu vorstellen sollte ist in der folgenden Grafik ersichtlich.

Der Zuhörer oder Konsument steht hier in einer Punkt-zu-Punktverbindung mit der musikerzeugenden Box. Diese strahlt die erzeugten Schallwellen radial ab, schliesslich erreichen diese Wellen das Ohr des Zuhörers. Er hört. Aber was hört er denn eigentlich? Er hört ausschliesslich den "Inhalt" der Schallwellen, sprich die Musik. Für räumliches Hören werden aber noch weitere Informationen benötigt. Eine dieser ist die Ortung der Musikquelle. Der Konsument hat

Abbildung 4.1: Mono Sound

es in diesem Fall recht einfach. Er kann die Quelle aufgrund winziger, zeitlicher Unterschiede der eintreffenden Schallwellen an den Ohren die Richtung bestimmen. Trifft das Signal erst am rechten Ohr, später am Linken ein, so liegt die Quelle rechts. Treffen die Schallwellen gleichzeitig ein, so befindet sich die Blickrichtung des Zuhörers genau auf der senkrechten Achse, die man zu den sich ausbreitenden Schallwellen vorstellen kann. Die Quelle muss also vor oder hinter ihm liegen, oder aber wenn es nicht so optimal läuft, ist die Quelle zwischen seinen Ohren.

4.2.2 Stereo

Die nächste Stufe ist hier der Stereosound. Meist ist er in der heimischen Musikanlage zu finden. Hier steht sich häufig ein Boxenpaar gegenüber, welches durch unterschiedlich Ansteuerung der einzelnen "Kanäle" eine Rechts-Linkseffekt erwirkt. Eine räumliche Ortung der verschiedenen Musikinstrumente ist durch den Hörer möglich, allerdings lässt er nur die Einordnung auf einer Linie zwischen den Boxenpaaren zu. Verschiedene Effekte wie etwa Hall oder die Veränderung der Frequenzen können hier zusätzlich eine veränderte Wirkung auf den Hörer erzielen (Die Einordnung des Zuhörers der Soundquellen auf der boxverbindenden Linie kann subjektiv als "weiter weg" empfunden werden). Hier eine kleine grafische Erläuterung.

Die Soundquelle kann also recht genau auf dieser virtuellen Linie geortet werden. Bei gut eingespieltem Sound von entsprechender Qualität sogar sehr genau. Es soll nicht unerwähnt bleiben, dass das menschliche Gehör eines unserer empfindlichsten Sinnesorgane darstellt. Den Vergleich zu einem Hund sollten wir dennoch scheuen. Andere Anforderungen haben andere evolutionäre Ergebnisse...

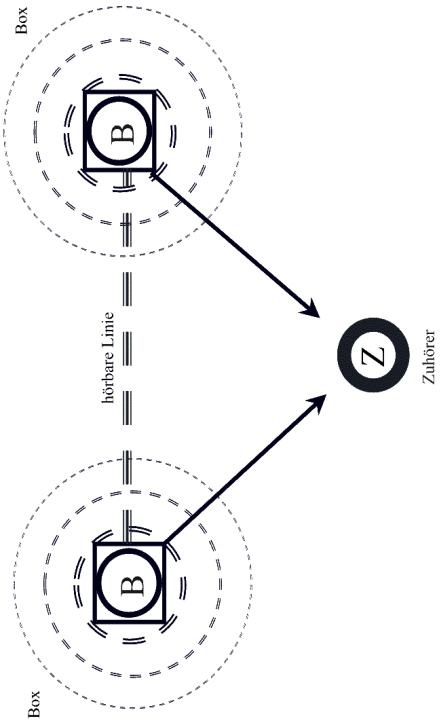

Abbildung 4.2: Stereo Sound

4.2.3 Dolby Surround

Die dritte oben aufgeführte Stufe ist der Dolby-Surround Sound. Dieser ist recht häufig in Kinos zu finden. Die Grundstruktur und Idee, die hinter diesem Konzept lässt sich folgendermaßen beschreiben. Um den Zuhörer werden Boxen auf einer Ebene installiert. Dies hat zur Folge, dass der Konsument das Gefühl eines "dreidimensionalen" Klangerlebnisses hat. Er kann Soundquellen in der Ebene orten, sprich vor sich und hinter sich sowie Kombination von rechts und links. Grafisch aufbereitet sieht dies ungefähr so aus :

Das für das Klangempfinden schon fast revolutionäre System hat für den Zuhörer viel Vorteile und lässt ihn sich tiefer an zum Beispiel einen Kinofilm fesseln. Das Erlebnis Kino findet nunmehr nicht nur auf der Leinwand statt, sondern erstmals auch hinter den Zuschauern. Flugzeuge kommen von "hinten" angeflogen. Man kann sie schon hören, obwohl sie im Film erst in den nächsten Sekunden auftaucht. Das gab es bei Stereosound auch, aber hier weiss man nicht nur ob es von links oder rechts kommt, sondern kann die Position genau bestimmen. Für den Zuhörer ein sehr eindrucksvolles Erlebnis.

4.2.4 Konzept des 3D-Sounds

Das Konzept unseres Soundraumes dagegen nimmt sich den zuletzt beschriebenen Dolby- Surround Sound als Grundstruktur und erweitert dieses Konzept. Die Erweiterung kann wie folgt beschrieben werden. Es werden praktisch 2 Dolby-Surround Systeme verwendet. Die Vorstellung, die dieser Idee zugrunde liegt ist aus der unten aufgeführten Grafik zu entnehmen.

Ein Raum, indessen Mittelpunkt sich der Zuhörer befindet hat 8 Ecken. Eine Dolby-Surround

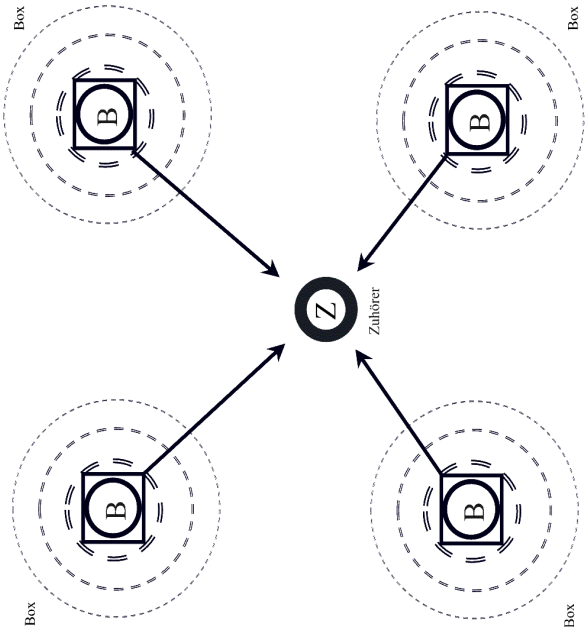

Abbildung 4.3: Dolby Surround

Anlage wird in der obenliegenden Ebene installiert, die andere in der unteren Ebene. Dadurch ist es möglich, dem Zuhörer ein richtiges dreidimensionales Klangerlebnis zu ermöglichen- bei einer korrekten Kopplung dieser beiden Systeme. Soundwege, also die Ortung der Klangquelle im Raum ist nun nicht nur in der Ebene möglich, sondern auch in der Höhe. Gegenüber Dolby-Surround ist dies eine wesentliche Verbesserung. Des weiteren war die Idee dieser Weiterentwicklung des Dolby-Surround das zusätzliche Einfügen von Effekten mittels der Soundkarte. Zusätzlich soll eine Art Frequenzweiche eingebaut werden. Diese soll den Musikstrom des Computers nach Frequenzen aufteilen und in Unterströmen ablegen. Diese Unterströme sollen auf die verschiedenen Boxen (insgesamt acht, eine in jeder Ecke des Raumes) gesendet werden. So lassen sich Effekte wie rotierende Bassläufe und so weiter realisieren.

Der Zuhörer soll im Endeffekt total in die technisch erzeugbare Klangwelt eintauchen und ein Klangempfinden wie in seiner natürlichen Umgebung haben. Man stelle sich hierzu eine Fliege vor, die den eigenen Körper umkreist. Mal oben, mal unten, mal rechts, mal links. Mit Dolby-Surround ist das nicht möglich, deswegen der 3D-Sound.

4.3 SoundRaum Inside

Der Sound Raum selbst gliedert sich in verschiedene Teile. Da wäre zum einen der Sound Server. Der Sound Server stellt der Aussenwelt eine CORBA Schnittstelle (siehe Abschnitt 4.3.1) zur Verfügung mit der man den Server plattformübergreifend sowie auch unabhängig von der verwendeten Programmiersprache ansprechen kann.

Die Sounds werden in einer speziellen Skript Datei definiert. Auf das Format der Skript Datei wird in Abschnitt 4.6 näher eingegangen. Zum Einlesen dieser Script Datei ist im Server ein Parser eingebaut, der dieses Dateiformat lesen kann und die dort definierten Aktionen und Definitionen an das eigentliche Modul zur Soundwiedergabe weiterleitet. Bei diesem Modul handelt es sich um eine Windows-Systembibliothek.

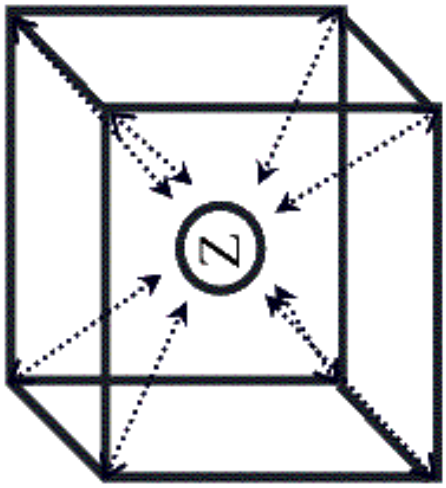

Abbildung 4.4: 3D Sound

Diese Systembibliothek namens SoundControl beinhaltet sämtliche Funktionalität zum Ansprechen der DirectX API (siehe auch unter Abschnitt 4.3.1) und zur Verwaltung der Sounds. Wir haben diese Lösung gewählt, da somit die eigentliche Funktionalität der Soundwiedergabe von der des Servers getrennt wird. Die Absicht dahinter war, dass man die Bibliothek später einfach in SNALG, unserem visuellen Editor zum erzeugen einer solchen Skript Datei, einbindet. So kann man dann bereits beim Erstellen der Skript Datei mit der Hilfe von SNALG die späteren Sounds originalgetreu abspielen. Diese DLL wird vom SoundServer einfach hinzugebunden, so das der eigentliche Server nichts mit der Soundwiedergabe zu tun hat.

Zum leichteren Erstellen eines solchen Sound Skriptes gibt es auch eine eigene GUI zum Ersparen des textuellen Editierens der Datei. Dieses Tool trägt den Namen SNALG (**S**ou**n**d **a**nd **L**ight **G**enerator. Licht deswegen, da der Name zu einer Zeit entstand, wo wir noch davon ausgingen, Licht in Abhängigkeit des Sounds zu schalten).

Die Abhängigkeit der verschiedenen Komponenten untereinander ist noch mal in Abbildung 4.5 Schaubild visuell verdeutlicht.

4.3.1 Verwendete Produkte

Bevor wir an die eigentliche Implementierung des SoundRaumes gingen mussten wir uns zuerst einmal für eine passende Soundkarte entscheiden. Zuerst hatten wir eine Soundkarte in der Preisklasse um 1000 DM im Auge gehabt. Diese Soundkarte hat den entscheidenen Vorteil, dass sie bereits von Natur aus 4 Stereo Ausgänge anbietet, mit deren Hilfe wir 8 Boxen hätten ansteuern können. Zum Wohle des finanziellen Gemütszustandes des Projektes haben wir uns dann aber nach einiger Evaluierungszeit für die Soundblaster Live von Creative Labs entschieden.

Diese Soundkarte fällt preislich entscheidend billiger aus (ca. 150 DM) stellt aber lediglich zwei Stereo Ausgänge zur Verfügung. Wir hatten damals den Gedanken im Hinterkopf eventuell zwei dieser Karten in einen Rechner zu installieren und diese dann zu koppeln, so das wir dann trotzdem 8 Boxen ansteuern können. Die konkrete Realisierung dieser Idee blieb jedoch aus,

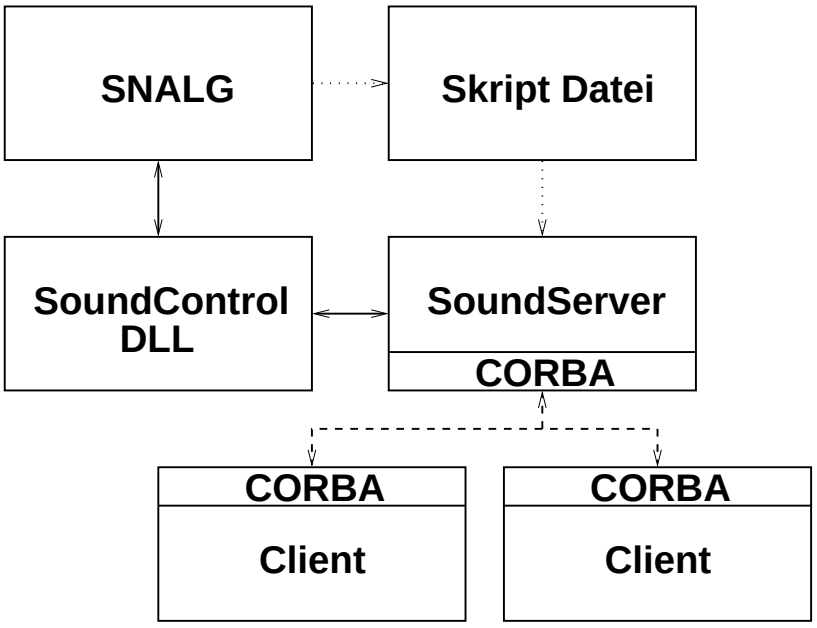

Abbildung 4.5: Komponenten des SoundRaumes

obwohl Implementierungstechnisch bereits Vorkehrungen zur Umsetzung vorhanden sind. Dies lag zum einen an dem immer knapper werdenden Zeitfenster, sowie der Tatsache, dass sich unter Windows nur einer dieser Karten im Rechner ansprechen lässt. Somit hätten wir zwei Windows Rechner mit Soundkarte gebraucht. Aufgrund der extremen Rechnerknappheit unseres Projektes kam dies somit nicht in Frage. Des weiteren käme es bei diesem Lösungsansatz zu Latenzzeit Problemen. Das menschliche Ohr ist was das Timing angeht ein ziemlich präzises Messinstrument. Latenzzeiten von wenigen Millisekunden hört man noch locker heraus. Die beiden Rechner müssten sich innerhalb der Latenzzeiten des menschlichen Gehörs gewissermassen absprechen. Da allein bei dem zugrundeliegenden CORBA Gerüst ein Methodenaufruf bereits um die 50 Millisekunden dauert haben wir uns von dieser Möglichkeit abgewandt und den SoundRaum lediglich mit 4 ansteuerbaren Boxen versehen.

Neben des Preises besitzt die Karte jedoch den weiteren Vorteil, dass man spezielle eingebaute Hardware Effekte über die allgemeine Geräuschkulisse legen kann. In der von Creative erhältlichen SDK zum Ansteuern der eingebauten EAX (Environmental Audio Extensions) Hardware gibt es bereits die in Abbildung 4.6 verewigten vordefinierte Effekte (wobei man auch hier seiner eigenen Kreativität beim Erstellen neuer Effekte völlig freien Raum lassen kann).

Da die Treiberunterstützung der verwendeten Soundkarte unter Windows zur damaligen Zeit (Anfang 2. Semester) wesentlich besser ausgebaut war, haben wir uns für diese Plattform entschieden. Ausserdem bestand unter Windows die Möglichkeit die DirectX API von Microsoft zu nutzen, welche unter anderem bereits rudimentäre Funktionen zum Abspielen von Sounds bereitstellt.

Als Entwicklungsumgebung haben wir den Visual C++ Compiler von Microsoft in der Version 6.0 benutzt. Dieser hat innerhalb der drei Semester andauernden Entwicklungsphase als recht problematisch herausgestellt. Doch mehr zu dieser Problematik in dem Abschnitt 4.7.2. Für die Programmiersprache C++ haben wir uns entschieden, da C/C++ für die verschiedensten Problemstellungen geeignet ist (obwohl es zweifelsohne nicht zu den syntaktisch schönsten Sprachen gehört). So kann man mit C++ sowohl auf einem relativ niedrigen Level arbeiten (ansprechen der Hardware), als auch auf einem recht hohen Level (unter anderem durch Verwendung der

Nr.	Effekt
0	Default
1	Gummizelle
2	Zimmer
3	Badezimmer
4	Wohnzimmer
5	versteinertes Zimmer
6	Zuhörersaal
7	Höhle
8	Arena
9	Hangar
10	Halle mit Tepichboden
11	Halle
12	Steinkorridor
13	Allee
14	Wald
15	Stadt
16	Berge
17	Steinbruch
18	Feld
19	Parkplatz
20	Kanalisation
21	Unterwasser
22	Besoffen
23	schwindlig
24	Psycho

Abbildung 4.6: Die vordefinierten EAX Umgebungseffekte

ebenfalls standardisierten Standard Template Library).

Um den Sound Server plattformübergreifend, sowie programmiersprachenunabhängig nutzen zu können, wurde er mit einer CORBA (Common Objects Request Broker Architecture) Schnittstelle ausgerüstet. CORBA ist ein von der OMG (Object Management Group) ausgearbeiteter Standard zur Kommunikation zwischen Objekten in objektorientierten verteilten Umgebungen- ähnlich Microsofts OLE/COM bzw. DCOM. Anders als Microsofts Standard ist es unter CORBA auch möglich plattformübergreifende Aufrufe zu tätigen.

Die implementierungsneutrale Beschreibung der Schnittstellen eines Server Objektes geschieht in der Interface Definition Language (IDL). Mittels eines IDL Compilers kann diese Schnittstellenbeschreibung in die meisten gängigen Programmiersprachen (z.B. C, C++, Perl, COBOL, etc) abgebildet werden. Die Syntax der IDL ist stark an diejenige von C++ angelehnt, wobei sie aber bei weitem keine komplette Programmiersprache in dem Sinne ist.

Mit Hilfe von CORBA kann man den Sound Server dann z.B. auch von Linux Rechnern aus ansprechen und steuern. Dies hielten wir zum Zeitpunkt der Entstehung für nötig, da wir von einer zentralen Steuerung des gesamten Projektes ausgingen die nicht zwingenderweise unter Windows geschehen musste. Als freien ORB haben wir uns nach einigem Testen für das unter GPL stehende MICO (Mico Is COrba) entschieden. Dieser ORB ist im Abschluß als recht positiv zu bewerten. Lediglich das Compilieren der MICO Sourcen unter Windows und MS Visual C++ gestaltete sich als zeit- und nervenfressendes Abenteuer. Aber dies ist wohl mehr auf Windows

zurückzuführen (siehe unter 4.7.2).

Für die Erstellung eines Parsers haben wir Flex und Bison (the tools formerly known as Lex and Yacc) verwendet. Mit Hilfe dieser beiden Tools lässt sich relativ schnell (nach entsprechender Einarbeitungszeit) ein Parser erstellen. Flex selber ist ein Tool zum erkennen von lexikalischen Ausdrücken. Bison dagegen ist ein Parser Generator, mit dessen Hilfe die von Flex erkannten Ausdrücke in einen grammatikalischen Kontext gestellt werden. Auch an dieser Stelle muss wieder über den Visual C++ Compiler hergezogen werden. So war es nicht möglich Bison unter Windows zum laufen zu kriegen. Glücklicherweise ist unsere Sound Description Language ja weit weniger anspruchsvoll als eine richtige Programmiersprache, so dass es sich letztendlich auch ohne Bison realisieren lies.

Zur Implementierung der eigentlichen Soundwiedergabe haben wir die DirectX (bzw. DirectSound) Entwicklungsumgebung von Microsoft in der Version 6.1 benutzt. DirectX ist eine API, die wohl in erster Linie für die Entwickler von Spielen erstellt wurde, damit diese eine wohldefinierte, hardwareunabhängige Schnittstelle zur Verfügung haben. So besteht DirectX auch aus verschiedenen Komponenten, wie z.B. DirectSound (die Schnittstelle zur Wiedergabe von Tönen), DirectPlay (zum Entwickeln von Netzwerkspielen), DirectDraw (2D Zeichenoperationen), DirectInput (Joysticks, Maus, Tastatur), Direct3D (dreidimensionales Rendering) und DirectSetup (zum Automatisieren der Installation). Die DirectX API ist relativ flexibel gehalten um den Spieleentwicklern keinen allgemeingültigen Standard vorzuschreiben. Dies bedeutet das die API auf einem relativ niedrigen Level gehalten ist und einem keineswegs die ganze Arbeit abnimmt (das Gegenteil ist eher der Fall, was zugegebenermassen auch eine vernünftiger Ansatz ist).

Beim Abspielen von MP3's haben wir uns vor einer Implementierung eines MP3 Decoders gedrückt und eine fertige SDK (XASDK) benutzt.

4.4 Die SoundControl DLL

Wie bereits erwähnt ist die SoundControl Bibliothek zuständig für das Abspielen, sowie das Verwalten (Position, Lautstärke, Abarbeiten der Befehlskette) der Sounds. Ein UML Diagramm der zentralen Klassen von SoundControl ist in Abbildung 4.7 abgebildet. In diesem Diagramm sind jedoch lediglich die zentralen Klassen abgebildet. So fehlen in diesem Diagramm z.B. die Klassen zum Log-System, zur Kontrolle der Wave Dateien oder die Hilfsklassen zur Abarbeitung der Befehlskette. Auf die nähere Beschreibung dieser Klassen verzichten wir da dort nichts sonderlich spannendes passiert und konzentrieren uns statt dessen auf den Kern des Ganzen.

Es gibt in SoundControl fünf verschiedene Klassen (GenericSoundBuffer und die vier Klassen die von GenericSoundBuffer erben) welche für die eigentliche Wiedergabe der Sounds zuständig sind. Die Aufteilung in fünf verschiedene Klassen ist vordergründig performanter Natur. So benötigt ein Buffer für dreidimensionale Sounds z.B. mehr Rechenzeit und Speicher als normale Sounds. Aufgrunddessen wird auch nur dann ein Buffer für 3D Sounds angelegt, wenn dies explizit gewünscht wird.

Die beiden Klassen SoundBuffer (Funktionalität für normale Sounds) und Sound3DBuffer (3D Sounds) werden abgeleitet von der Klasse StreamBuffer. Da man in der Regel nur begrenzt Hauptspeicher zur Verfügung hat befindet sich immer nur ein Teil der Sounddaten im Buffer. Dieser Buffer ist in unserem Fall groß genug für die Wiedergabe von zwei Sekunden. Die DirectX API ist auf einem relativ niedrigen Niveau besiedelt, so das man sich um die Erneurung der Daten im Buffer selbst kümmern muss. Dies geschieht in der StreamBuffer Klasse. Somit ist

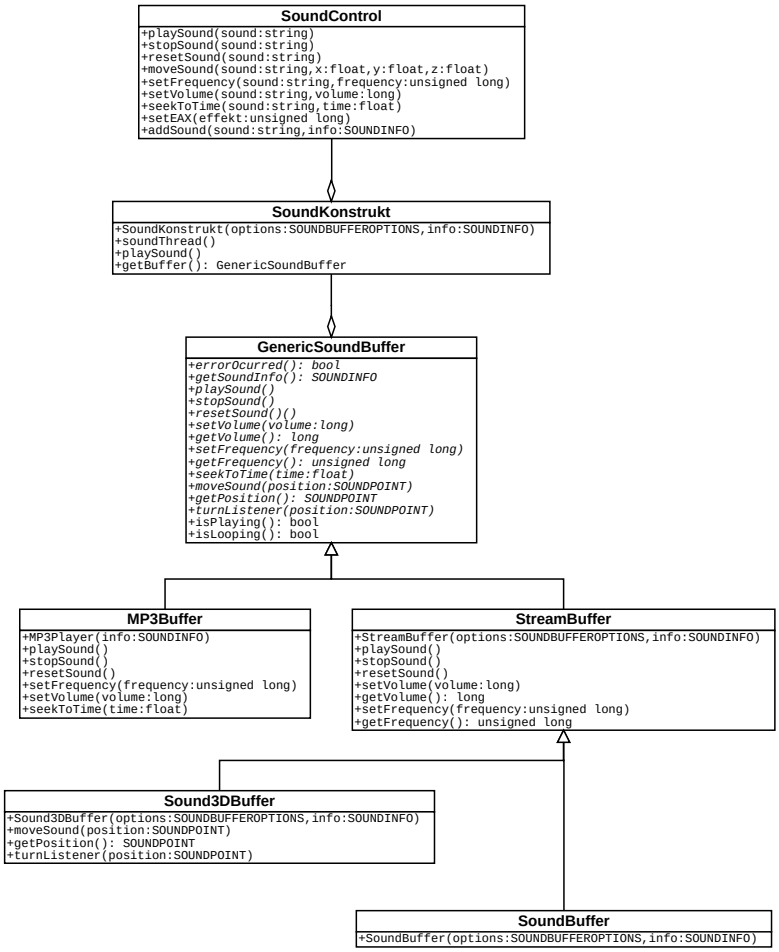

Abbildung 4.7: Klassenstruktur innerhalb von SoundControl

sämtliche Logik zum Ansteuern der DirectX Schnittstelle lediglich in den Klassen StreamBuffer, SoundBuffer und Sound3DBuffer enthalten (mal abgesehen vom Initialisieren der DirectSound Schnittstelle).

In der MP3Buffer Klasse ist die Funktionalität zum Abspielen von MP3 Dateien. Das eigentliche Abspielen geschieht mittels der XASDK API.

Die gemeinsame Oberklasse für alle Arten von Sounds ist die Klasse GenericSoundBuffer. Dort werden die Informationen gehalten, die für alle anderen Buffer von Bedeutung sind. Dies wären z.B. Dateiname, Lautstärke, Frequenz oder der Status (wird der Sound gerade abgespielt). Des weiteren stellt die Klasse leere, virtuelle Methodenrümpfe zur Verfügung, welche die abgeleiteten Klassen je nach Bedarf, mit ihrer eigenen Funktionalität überschreiben (z.B. überschreiben MP3Buffer, SoundBuffer oder StreamBuffer nicht die Methode moveSound(). Sound3DBuffer jedoch schon).

Die Funktionalität zum Abarbeiten, der in der Skript-Datei definierten Befehlskette, ist in einer weiteren Klasse namens SoundKonstrukt enthalten. Beim Abspielen eines Sounds wird ein neuer Thread gestartet. Dieser Thread arbeitet die Befehlskette, sofern sie in der Skript Datei definiert wurde, Stück für Stück ab und leitet die Befehle entsprechend an eine Instanz von GenericSoundBuffer weiter. Hier kommt die vorher besprochene Klassenstruktur zum Tragen. Da alle SoundBuffer generell von GenericSoundBuffer erben kann es sich hierbei um ein SoundBuffer, Sound3DBuffer oder StreamBuffer handeln.

In der Praxis verwaltet die SoundControl DLL die ganzen Sounds in einer Hashmap, die instanzierte SoundKonstrukt Objekte beinhaltet. Indiziert werden die Objekte über die symbolischen Namen, die in der Skript Datei definiert wurden. Wenn nun ein Befehl für Sound xyz ankommt, so überprüft SoundControl, ob in seiner Hashmap ein Eintrag für xyz vorhanden ist. Ist dies der Fall, so wird das SoundKonstrukt Objekt anhand des Schlüssels xyz geholt und der Befehl an dieses Objekt weitergeleitet.

Die SoundControl Klasse stellt auch gleichzeitig die Schnittstelle zur eigentlichen Bibliothek dar. Die Methoden dieser Klasse sind alle gleichbedeutend mit der im folgenden Kapitel vorgestellten Funktionalität der IDL Schnittstelle. Einzig die Methode addSound() taucht nicht in der IDL Schnittstelle auf. Mit dieser Methode wird ein neuer Sound hinzugefügt. Dieser Aufruf geschieht durch den eigentlichen Server nach dem dieser eine gültiges Sound Statement in der Skript Datei gefunden hat.

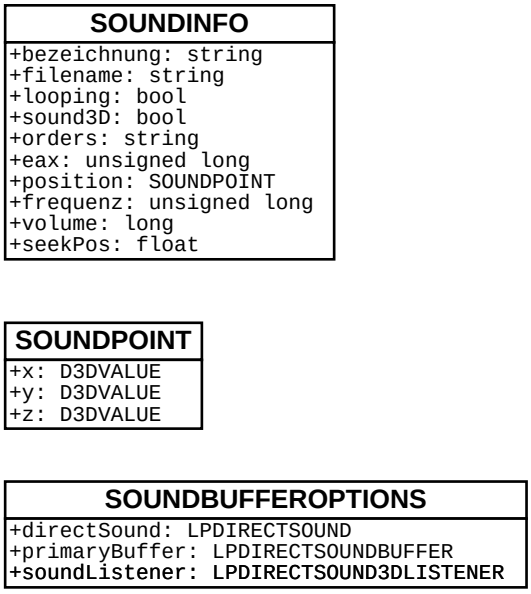

Abbildung 4.8: Interne Datenstrukturen

In der Abbildung 4.8 sind die zentral verwendeten Datenstrukturen abggebildet. Die Struktur SOUNDPOINT ist die Position im dreidimensionalen Raum. Referenzen auf die bereits Initialisierten DirectX Schnittstellen werden in der Struktur SOUNDBUFFEROPTIONS gespeichert. Den Zweck der einzelnen Variablen näher zu beschreiben würde zu sehr in die Internas hineinsteigen. Diese Struktur wird lediglich von den verschiedenen SoundBuffern gebraucht um sich selbst zu initialisieren.

Da der Datentyp SOUNDINFO als Argument der Methode addSound() der Klasse SoundControl erwartet wird, haben wir die genaue Bedeutung der Variablen in der Abbildung 4.9 abgebildet. In dieser Datenstruktur werden die kompletten Daten über einen Sound gehalten, welche für die Applikation relevant sind.

4.5 Der SoundServer

Der SoundServer stellt eine über CORBA ansprechbare Schnittstelle zur Verfügung. Über diese Schnittstelle kann man die definierten Sounds z.B. starten oder stoppen. Wir haben auch die Möglichkeit vorgesehen gewisse Parameter (Lautstärke, Effekt, Frequenz, usw) zur Laufzeit zu

Attribut	Bedeutung
bezeichnung	Der in der Skript Datei definierte Name des Sounds
filename	Die Datei mit den Sound Daten (MP3 oder WAV)
looping	Soll der Sound immer wieder von vorne wiederholt werden?
sound3D	Handelt es sich um einen 3D Sound?
orders	die komplette Befehlskette
eax	Der Effekt
position	Die gegenwärtige Position im Raume
frequenz	Die Abspielfrequenz
volume	Die Lautstärke
seekPos	Die gegenwärtige Abspielposition

Abbildung 4.9: Die nähere Beschreibung der Attribute von SOUNDINFO

verändern. So besteht einerseits die Möglichkeit einen festen Ablauf mit Hilfe der Skript Datei zu definieren und andererseits kann man durch die Schnittstelle des Servers eine gewisse Interaktivität realisieren. Bei der eigentlichen Vorführung kamen die interaktiven Aspekte jedoch nicht zum Tragen.

Beim Start des Servers versucht dieser automatisch eine Skript Datei mit dem Namen script.snd zu laden. Beim Parsen einer gültigen Sound Anweisung wird eine Struktur vom Typ SOUNDINFO mit den entsprechenden Initialwerten erstellt. Die komplette Befehlskette, sofern natürlich vorhanden, wird ebenfalls in dieser Struktur gespeichert. Diese Struktur wird dann an die addSound() Methode der Klasse SoundControl übergeben.

Aufgrund der Kürze des Source Codes und der Tatsache das dies, die für die Aussenwelt sichtbare Schnittstelle ist, haben wir die CORBA Schnittstelle in Abbildung 4.10 verewigt. Der Code spricht größtenteils für sich selbst.

Es gibt eine Methode startSound() zum Abspielen eines Sounds und eine Methode stopSound() zum Stoppen des Selbigen. Das string Argument bezeichnet hierbei bei allen Methoden (Aussnahme setEAX) den Namen des Sounds für den die Methode ausgeführt werden soll. Dieser Name ist frei wählbar und muss lediglich in der Skript Datei definiert sein. Mit moveSound() bewegt man den Sound zu einer dreidimensionalen Koordinate. Die Y-Koordinate ist generell beim SoundRaum unbenutzt, da wir ja in der endgültigen Version lediglich 2 Ausgänge auf der Soundkarte haben (4 Boxen) und somit keine richtigen 3D Sounds produzieren können. Sie ist lediglich für eventuelle Erweiterungen vorhanden.

Mittels setFrequency() kann man die Frequenz eines Sounds während der Laufzeit ändern. Die Integer Variable gibt dabei die neue Frequenz an. Ditto für setVolume(), nur das hier die Lautstärke verändert wird. Durch seekToTime() kann man die Abspielposition des SoundBuffers verändern. Die Gleitkommazahl gibt an, zu welcher Position (in Sekunden ab Anfang) wir uns bewegen.

Die große Ausnahme bei den Methoden ist setEAX(). Hier wird kein Soundname angegeben, da die EAX Effekt grundsätzlich auf den gesamten Sound drübergelegt werden. Die Integer Variable gibt den zu nutzenden Effekt an. Für eine Liste der vordefinierten EAX Effekt siehe auch Abbildung 4.6.

```
//! IDL Interface für den SoundServer.
interface SoundRaum
{
    // Sound abspielen
    void playSound(in string sound);

    // Sound stoppen
    void stopSound(in string sound);

    // Sound reset (wieder auf Anfang zurück)
    void resetSound(in string sound);

    // Sound bewegen
    void moveSound(in string sound,
                   in float x,
                   in float y,
                   in float z);

    // Frequenz ändern
    void setFrequency(in string sound,
                      in unsigned long freq);

    // Lautstärke ändern
    void setVolume(in string sound,
                   in long volume);

    // Bufferposition des Sounds verändern
    void seekToTime(in string sound,
                    in float seconds);

    // Effekt ändern
    void setEAX(in unsigned long eax);
};
```

Abbildung 4.10: CORBA Schnittstelle zum SoundServer

4.6 Die Sound Description Language

Bevor wir an die Beschreibung unserer formalen Sound Sprache näher eingehen, wollen wir zuerst einmal auf das generelle definieren von Sounds mit Hilfe der Sound Description Language eingehen.

In der Skript Datei werden frei wählbare, symbolische Namen für die verschiedenen Sound Ereignisse definiert. Diesen definierten Sounds kann man noch verschiedene Aktionen zuordnen, welche ebenfalls in der Skript Datei definiert werden. Generell kann man hier zwischen zwei verschiedenen Arten von Aktionen unterscheiden. Zum einen kann man dem Sound verschiedene Initiale Parameter zuordnen wie z.B. Lautstärke, Frequenz, Anfangsposition oder gewünschter Effekt.

Des weiteren besteht die Möglichkeit gewünschte Laufzeitaktionen zu definieren. Dies wird in einer Befehlskette näher definiert. Man hat die Möglichkeit ein Ereigniss zu einem bestimmten Zeitpunkt auszulösen was über eine gewisse Dauer gehen soll. Die Zwischenergebnisse werden interpoliert. Die Angabe der Ereignisse muss keinesfalls in chronologischer Reihenfolge geschehen (obwohl dies der Lesbarkeit eher hilfreich ist) da die Befehlskette intern wieder in die richtige Reihenfolge gebracht wird. Die Ereignisse, denen man eine gewisse Dauer zuordnen kann wären: Frequenz ändern, Lautstärke ändern sowie Position verändern (siehe auch in der formalen Grammatik).

Die Grammatik für die Sound Description Language (auf das Wesentliche vereinfachte Version) ist in der Abbildung 4.11 wiedergegeben. Alle terminalen Symbole sind durch Hochkommata kenntlich gemacht.

4.6.1 Beispiele

Die folgenden zwei Beispiele sollen die oben definierte Grammatik etwas plastischer darstellen. Die Beispiele sind aus dem echten Skript zur Präsentation entnommen worden.

```
sound mies
{
  frequence = 44100;

  filename = "d:\sounds\mies.wav";

  orders = {
        (at 00:02 stop)
        (at 00:03 play)
        (at 00:05 stop)
        (at 00:06 play)
        (at 00:08 stop)
        (at 00:09 play)
  }
} sound
```

Dies ist das Techno Gewummer welches im Verlaufe der Vorführung irgendwann ertönt. Diesem Sound wird die Bezeichnung “mies” zugeordnet. Anhand dieser Bezeichnung kann der SoundRaum den Sound jederzeit finden. Die Frequenz mit der dieser Sound abgespielt werden soll ist auf 44100 Hertz eingestellt. Als Wave Datei wird “mies.wav” benutzt. Zu den definierten

```
START        = SOUND START | ''

SOUND        = OPENSOUND FILENAME INSTRUCTIONS ORDERS CLOSESOUND
OPENSOUND    = 'sound' IDENT '{'
CLOSESOUND   = '} sound'

INSTRUCTIONS = INITEAX INSTRUCTIONS |
               INITPOS INSTRUCTIONS |
               INITFREQ INSTRUCTIONS |
               INITVOL INSTRUCTIONS |
               INITLOOP INSTRUCTIONS | ''

INITEAX      = 'eax = ' NUMBER ';'
INITPOS      = 'pos = ' VECTOR ';'
INITFREQ     = 'frequence = ' NUMBER ';'
INITVOL      = 'volume = ' INTEGER ';'
INITLOOP     = 'looping;'


ORDERS       = 'orders = {' SINGLEORDER+ '}'

SINGLEORDER  = '(at ' TIME (POSCHANGE | FREQCHANGE | EAXCHANGE |
               VOLCHANGE | 'stop' | 'play' | 'reset') ')'

POSCHANGE    = 'moveto ' VECTOR ('in' NUMBER)?
FREQCHANGE   = 'frequence ' NUMBER ('in' NUMBER)?
EAXCHANGE    = 'eax ' NUMBER
VOLCHANGE    = 'volume ' INTEGER ('in' NUMBER)?

FILENAME     = 'filename = "' (LETTER | DIGIT | SPECIAL)+ '";'
IDENT        = LETTER (LETTER | DIGIT)*
LETTER       = 'a'-'z' | 'A'-'Z'
DIGIT        = '0'-'9'
NUMBER       = DIGIT+
INTEGER      = ('-' | +')? DIGIT+
FLOAT        = INTEGER | INTEGER '.' DIGIT+
SPECIAL      = '-'|'+'|'\'|':'|'.'|'_'|'/'
VECTOR       = '[' FLOAT FLOAT FLOAT ']'
TIME         = NUMBER ':' NUMBER
```

Abbildung 4.11: Die formale Grammatik für die Sound Description Language

Zeitpunkten wird der Sound für eine Sekunde gestoppt und dann weitergespielt.

```
sound wasserwind
{
  looping;

  volume = -3000;
  filename = "d:\sounds\wasserwind.wav";

  orders = {
        (at 00:01 volume 0 in 13)
  }
} sound
```

Bei diesem Sound wurde die Initiallautstärke auf sehr leise eingestellt (-3000). Diese Lautstärke wird jedoch ab Sekunde Eins des Abspielens langsam auf 0 (normal) erhöht. Dieses Einfaden soll innerhalb von 13 Sekunden geschehen.

Obwohl die bereits gezeigten Beispiele recht einfacher Natur sind, kann man natürlich auch weitaus komplexere Befehlsketten definieren.

4.7 SNALG

Zur schnellen und einfachen Erstellung der benötigten Skriptdateien ist die Umsetzung eines Skripteditors nötig. Dieser wird im Folgenden SNALLG genannt, was für Sound'n Light Language Generator steht. Die Nichtumsetzung, bzw. die Ausgliederung der gesamten Lichtsteuerung hatte hier keine Auswirkungen auf die Namensgebung.

4.7.1 Funktionsweise und Beschreibung

Der Soundeditor SNALLG hat eine grafisch orientierte Benutzeroberfläche. Hier kann der Anwender schnell und einfach die benötigten Soundfiles (.wav und .mp3) auswählen und in das Skript einfügen.

Jeder Soundfile kann mit verschiedenen Effekten belegt werden. Die Liste der Effekte taucht an anderer Stelle dieser Dokumentation auf und wird hier nicht nochmals wiederholt. Auch verschiedene Effekte lassen sich gleichzeitig anwählen. Die Anzahl ist theoretisch nicht begrenzt.

Des weiteren besteht die Möglichkeit einen Soundweg zu definieren. Hierzu wird eine 3-dimensionaler Raum angezeigt, der beliebig drehbar ist. Jede Ecke repräsentiert hierfür eine Box. Der Anwender hat nun die Möglichkeit einen "Weg" in diese Grafik einzuzeichnen. Dieser wird später vom Soundserver umgerechnet und versorgt die benötigten Boxen mit den neu errechneten Audioströmen. Die Implementation ist allerdings nicht vollständig geglückt, was auf den erheblichen Zeitmangel zurückzuführen ist.

Weitere Editiermöglichkeiten sind das Anheben der Lautstärke, sowie das Senken derselben.

Alle Ereignisse sind zeitgesteuert, das heißt, man hat die Möglichkeit, einen genauen Zeitpunkt festzulegen, an dem der gewünschte Effekt in Aktion tritt.

Die somit festgelegten Eigenschaften eines einzelnen Soundfileobjektes lassen sich im Editor speichern. Man kann nun weitere Files hinzufügen und mit weiteren Effekten belegen.

Hat man alle erforderlichen Eingaben gemacht, ist der Editor fertig zum Kompilieren des Skriptes. Hier wird intern eine Interpretation der Eingaben vorgenommen und formatiert ausgegeben. Das Skript kann an einem vorher festgelegten Speicherort abgelegt werden, was Vorteile hinsichtlich der Bedienbarkeit mit sich bringt. Der Soundserver kann diesen Ort als Standardposition nehmen und verhindert somit das umständliche Kopieren der Datei.

Auch die vom jeweiligen Skript benötigten Dateien, welche die Audioströme enthalten werden an die Orte des Verzeichnisbaumes kopiert, wo sie vom Soundservers erwartet werden.

4.7.2 Aufgetretene Probleme

Bei der Umsetzung des Sound Raumes und dessen Implementierung sind wir auf zahlreiche Probleme gestossen. So hatte vor allem Windows uns ein paar beträchtliche Probleme zugefügt. Wir haben uns für Windows als Plattform entschieden, da zu der damaligen Zeit (2. Semester) die Treiberunterstützung für die Soundkarte unter Windows besser war. Die Treiberunterstützung unter Linux ist im Laufe des Projektes jedoch entschieden besser geworden, so das dieses Argument hat im Verlaufe des Projektes hinfällig geworden ist.

Besonders zu schaffen hat der MS Visual C++ Compiler von Microsoft gemacht. Hier hat Microsoft mal wieder versucht das Rad neu zu erfinden und hat ihre eigenen Standards gemacht. In Anbetracht der Tatsache, dass es bereits einen eigenen verabschiedeten C++ Standard gibt, ist dies völlig unverständlich. Die Microsoft Erweiterungen lassen sich zwar ausschalten, jedoch hat man dann Probleme beim hinzubinden der Systembibliotheken.

Aufgrund dessen hat es zu einigen Problemen beim nutzen von anderen Produkten geführt. So war es zwar unter Windows möglich den lexikalischen Analyzer Flex zu benutzen, bei der Unterstützung des Parsergenerators Bison hat es trotz stärkster Bemühungen diesen zum Laufen zu bringen jedoch aufgehört. Da unsere Sprache zur Beschreibung der Sounds jedoch weit hinter einer "echten" Programmiersprache hinterherhinkt war es uns möglich den Parser nur mithilfe von Flex zu implementieren. Die wenige Logik haben wir dann "von Hand" implementiert.

Auch die Suche nach einem passenden, frei verfügbaren, ORB war unter Windows eine kleine Odysee. So stehen zwar einige freie ORBs zur Verfügung, die Auswahl derjenigen die aber auch unter Windows nutzbar sind ist aber ziemlich gering. Orbacus ist z.B. ein relativ guter und schneller ORB. Dummerweise lies dieser sich trotz mehrtägiger Versuche nicht unter Windows compilieren. Am Ende sind wir dann bei dem frei verfügbaren MICO gelandet. Dieser lies sich nach einigen Problemen auch unter Windows nutzen.

4.8 Fazit

Wenn man das eigentliche Endergebnis (sprich Aufführung) sieht, welche ja im Wesentlichen einen festen Ablauf und wenig Interaktivität beinhaltet, so müssen wir feststellen das wir einiges gemacht haben was schlichtweg überflüssig ist. Dies liegt aber vor allem daran, dass sich die eigentlichen Projektformen erst kurz vor Ende des Projektes herauskristallisiert haben (durch die Zusammenarbeit mit Martin und Jörg Richard). Wir gingen ansich davon aus, das das Endprodukt einen ziemlich großen Anteil von Interaktivität beinhaltet und haben deswegen auch Möglichkeiten vorgesehen um dies umzusetzen. Hätte man bereits am Anfang gewusst in welche

Richtung das Projekt am Ende schlägt, so hätten wir uns beim SoundRaum wesentlich mehr auf das Notwendige konzentriert.

Viele Möglichkeiten des SoundRaumes blieben ungenutzt. Im Wesentlichen wurden einfach die WAV Dateien wiedergegeben. Effekte, die Möglichkeit zur Laufzeitinteraktion, die Nutzung von vordefinierten Befehlsketten (ausser zum Ein- bzw. Ausfaden) und andere Sachen blieben größtenteils ungenutzt. Dies lag zum einen an den fest definierten Ablauf und zum anderen blieb am Ende nicht mehr viel Zeit um die verschiedenen Sounds richtig auszutüfteln. Da wir ja von einer gewissen interaktiven Steuerung des SoundRaumes ausgingen, haben wir einen kompletten CORBA Server entworfen. Diese Möglichkeiten wurden jedoch überhaupt nicht genutzt, so dass dieser schlichtweg überflüssig ist (zumindest bei der Aufführung).

Nichtsdestotrotz (oder gerade deswegen) war der Lerneffekt dieses Projektes immens. Wir haben in vielerlei Sachen Neuland betreten. So hatten wir uns vorher in keinster Weise mit dem Erstellen eines Parsers, CORBA, Flex & Bison, DirectX oder Windows Programmierung beschäftigt.

Kapitel 5

Lichtsteuerung

5.1 Einleitung

Die Beleuchtung einer Bühne dient mehr als nur dem Ausleuchten zum Zwecke des Sehenskönnens. Mit Licht haben Theaterregisseure ein Mittel zur Verfügung, mit relativ wenig Aufwand die Ausdruckskraft der Kulissen, Requisiten und Darsteller zu verändern. Es ist möglich, nur durch Licht die Aufmerksamkeit des Publikums von einem bestimmten Punkt oder auf einen bestimmten Punkt abzulenken. Auch kann man die Bühne in verschiedene Spielwelten einteilen, ohne jedes Mal umbauen zu müssen, halb durchsichtige Zwischenvorhänge aus Gaze können nur durch Licht fast völlig transparent werden oder auch absolut undurchsichtig. Man kann mit Gegenlicht, Streiflicht oder Schatten arbeiten. Durch Dutzende Sorten Farb- oder Effektfilter kann man den Zuschauern Empfindungen suggerieren, angefangen bei Behaglichkeit bis zur extremen Ungemütlichkeit bzw. Kälte. Alle diese Mittel können dem Publikum sowohl direkt wahrnehmbar als auch unbewusst dargeboten werden, d.h. beispielsweise durch einen sehr langsamen Lichtwechsel von einem warmen, weichen Bühnenlicht zur kalten, gleißenden Helle kann eine Handlung unterstützt werden, bei der den Charakteren des Stückes z.B. nach und nach ein Problem über den Kopf wächst. Durch Farben und andere Lichteffekte können aber auch Tages- oder Jahreszeiten, Wetter oder Feuer dargestellt werden.

Wegen dieser Möglichkeiten wird der Beleuchtung von den Regisseuren auch große Beachtung geschenkt. Das sogenannte Einleuchten ist der Vorgang, bei dem die benötigten Scheinwerfer ihrer Funktion im Stück zugeordnet, auf die richtige Position ausgerichtet und mit Farbscheiben bestückt werden. Dieses Einleuchten muss selbstverständlich auf jeder Bühne oder nach jeder Spielpause wieder neu durchgeführt werden. Beim ersten Einleuchten wird oftmals lange Zeit mit verschiedenen Lichteffekten experimentiert, entweder hat der Regisseur bereits vorher feste Vorstellungen, wie die Szenen aussehen sollen, oder er probiert mit den Beleuchtern verschiedene Möglichkeiten durch. Irgendwann, wenn die Beleuchtung fest steht, werden die verschiedenen Einstellungen, in Fachkreisen Stimmungen genannt, zu Papier gebracht, oder falls man ein Computerlichtpult hat, dort gespeichert. Dabei werden auch die verschiedenen Umblendzeiten erfasst. Um das Aufschreiben, bzw. gegebenenfalls Ausdrucken, kommt man zumindest dann aber kaum herum, wenn man auf Tournee gehen will, denn kein Theater ist wie das andere; selbst wenn man den gleichen Computer vorfindet, so sind andere Scheinwerfer vorhanden.

5.2 Aufgabenstellung

Die Lichtsteuerung ist für die Bühnenbeleuchtung auf der Bühne zuständig. Durch die Lichtsteuerung können anhand einer Relais-Karte Lichter einzeln angesteuert und auch mehrere Lichter gleichzeitig gesteuert werden. Es können auch mit Hilfe der Lichtsteuerung Sequenzen gespeichert und abgespielt werden.

5.3 Relaiskarte

Die Ansteuerung der Lichtanlage wird über eine modifizierte Ein-/Ausgabe-Karte realisiert. Die Karte verfügt in der Grundausstattung entweder über acht Eingänge oder über acht Ausgänge. Die Eingänge werden über Optokoppler galvanisch getrennt. Die Ausgänge werden über Relais geschaltet, die über Leistungstreiber angesteuert werden.

Die Original-Karten wurden modifiziert, da sie entweder nur als Aus- oder Eingabe-Version bestückt waren. Um alle Ein- und Ausgänge beider Karten an einen Druckerport benutzen zu können, wurden einige Funktionen mittels Programmierbaren Logikbausteinen(GAL) der Firma Lattice realisiert.

Die Beschreibungsdatei für den Ausgabespeicher der Karte sieht wie folgt aus:

```
title    Latch 8 x
pattern  Out8x
revision A
author   Denis Boehme
company  Projekt THEMA 1998-2000 Uni-Bremen
date     6.12.1998

chip Out8x gal16v8

; pin   1    2    3    4    5    6    7    8    9     10
        clk  i0   i1   i2  state id0 id1   a0  a1    gnd

; pin   11   12   13   14   15   16   17   18   19    20
        /en  o1   o2   o3   o4   o5   o6   o7   o8   vcc

@ues Out11x

;
;  Relais-Ansteuerung
;
;
; Version V2
;
; Diese Version speichert die Zustände im GAL.
;
equations

o1   :=  state * /i0 * /i1 * /i2 * /id0 * /id1 * /a0 * /a1
     +    i0   *  o1
     +    i1   *  o1
     +    i2   *  o1

o2   :=  state *  i0 * /i1 * /i2 * /id0 * /id1 * /a0 * /a1
     +   /i0   *  o2
     +    i1   *  o2
     +    i2   *  o2

o3   :=  state * /i0 *  i1 * /i2 * /id0 * /id1 * /a0 * /a1
     +    i0   *  o3
     +   /i1   *  o3
     +    i2   *  o3
```

```
o4   :=  state *  i0 *  i1 * /i2 * /id0 * /id1 * /a0 * /a1
     +   /i0   *  o4
     +   /i1   *  o4
     +    i2   *  o4

o5   :=  state * /i0 * /i1 *  i2 * /id0 * /id1 * /a0 * /a1
     +    i0   *  o5
     +    i1   *  o5
     +   /i2   *  o5

o6   :=  state *  i0 * /i1 *  i2 * /id0 * /id1 * /a0 * /a1
     +   /i0   *  o6
     +    i1   *  o6
     +   /i2   *  o6

o7   :=  state * /i0 *  i1 *  i2 * /id0 * /id1 * /a0 * /a1
     +    i0   *  o7
     +   /i1   *  o7
     +   /i2   *  o7

o8   :=  state *  i0 *  i1 *  i2 * /id0 * /id1 * /a0 * /a1
     +   /i0   *  o8
     +   /i1   *  o8
     +   /i2   *  o8
; end of Out8x
```

Diese Bausteine mussten mit Hilfe von speziellen Programmiergeräten programmiert werden.

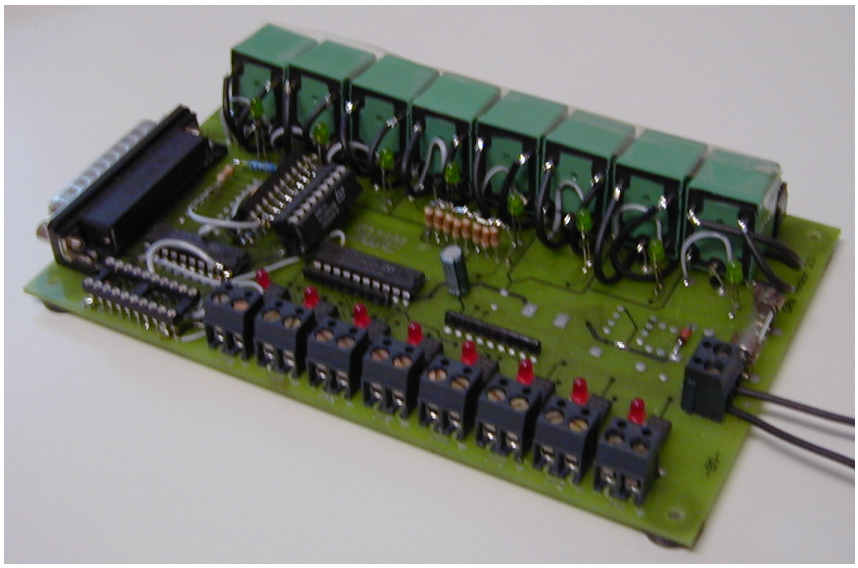

Abbildung 5.1: Die umgebaute Relais-/Opto-Koppler-Karte

Nach der Modifizierung konnten diese Karten sowohl für Eingabe-, als auch für Ausgabezwecke verwendet werden. Die Schaltzustände der Ausgänge wurden anfangs auf der Karte gespeichert. Im praktischen Betrieb mit starken Verbrauchern an den Ausgängen, stellten sich jedoch Störungen und instabile Schaltzustände ein. Selbst Filter brachten hier keine Abhilfe, so dass die Programmierung der GALs geändert wurde. Nach dieser Änderung wurden wieder

direkt die Schaltzustände der Datenleitungen des Druckerports an die Relais geleitet. Störungen traten nicht mehr auf, dafür war jetzt nur noch eine Karte pro Druckerport benutzbar. Dies war für die vorgesehene Anwendung jedoch ausreichend.

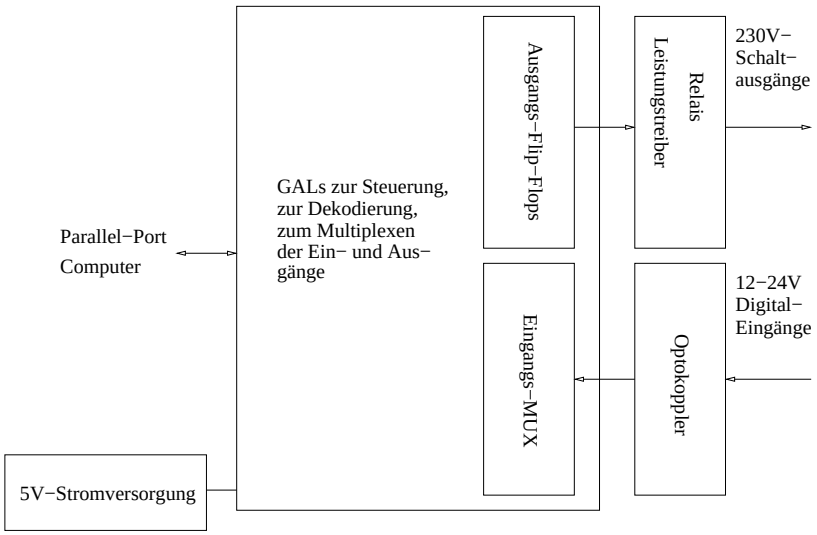

Abbildung 5.2: Eine schematische Darstellung der Relais/Opto-Koppler-Karte

5.4 Server-Software

Der Server, der die Relaiskarte ansteuert, wurde unter Linux realisiert. Es handelt sich um einen TCP/IP-Socket-Server, der nur einen Klienten zur Zeit zulässt.

Bei der Implementierung wurde die APE-C++-Library verwendet, welche später in die CommonC++-Library aufgenommen wurde. Diese Bibliothek soll es ermöglichen Plattformunabhängige Server-Programme in C++ zu implementieren. Als Implementierungssprache wurde demnach C++ verwendet.

5.5 Client-Software

Das Programm zur clientseitigen Ansteuerung der Lichtanlage wurde unter Microsoft Windows 98 realisiert.

5.5.1 Systemumgebung

Die Lichtsteuerung ist als Delphi-Applikation implementiert. Durch die Implementierung mit einer GUI ist eine grafische Benutzungsoberfläche Voraussetzung. Die Lichtsteuerungs-GUI wurde unter Windows 98 realisiert.

5.5.2 Benutzer/ Bediener

Die Lichtsteuerung ist leicht zu verstehen und der Umgang ist auch relativ einfach. Der Benutzer sollte grundlegende Kenntnisse über Netzwerkkommunikation haben, da die Lichtsteuerung darüber kommuniziert.

5.5.3 Anforderungen

Die Lichtsteuerung soll mehrere Aufgaben erfüllen:

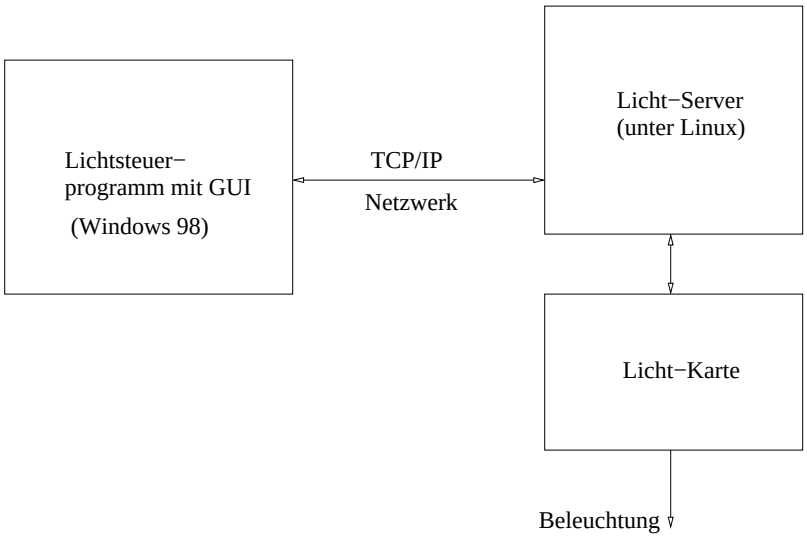

Abbildung 5.3: Zusammenspiel des Servers, Clients und der Relaiskarte

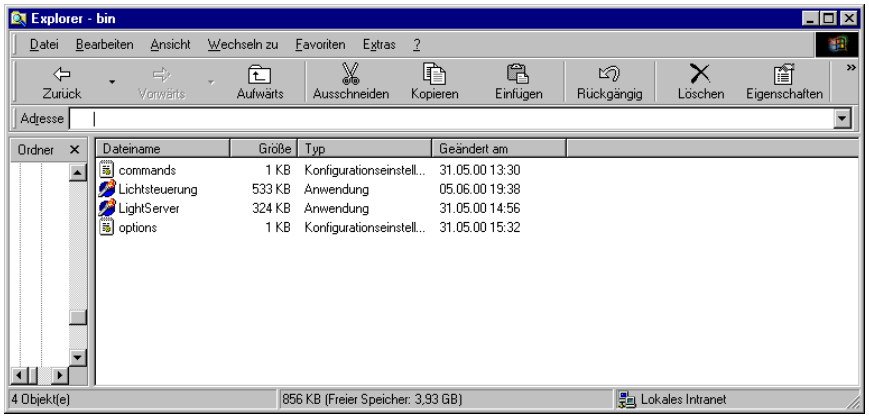

Abbildung 5.4: Durch anklicken der Lichtsteuerung wird unsere GUI gestartet

Monitorfunktion: Die Monitorfunktion soll folgende Funktionalität bieten:

- Der Anwender soll die Aktivitäten der Lichtsteuerung überwachen können.
- Welches Licht ist gerade an? Bin ich mit dem Lichtserver verbunden?
- Welche Sequenzen habe ich gespeichert?

Benutzungsschnittstelle: Dem Benutzer sollen verschiedene Funktion zur Verfügung gestellt werden:

- Die Lichtsteuerung muß problemlos an den Lichtserver anbindbar sein.
- Man soll ohne Probleme die Lichtsteuerung starten und beenden können.
- Eine Protokolldatei soll kreiert werden.

5.5.4 Plattform- und Standortunabhängigkeit

Aus Gründen der bequemeren Nutzung, soll die Lichtsteuerung auf so viele Plattformen wie möglich portierbar sein. Es soll möglich sein z.B. von einem Windows- PC auf den Lichtserver

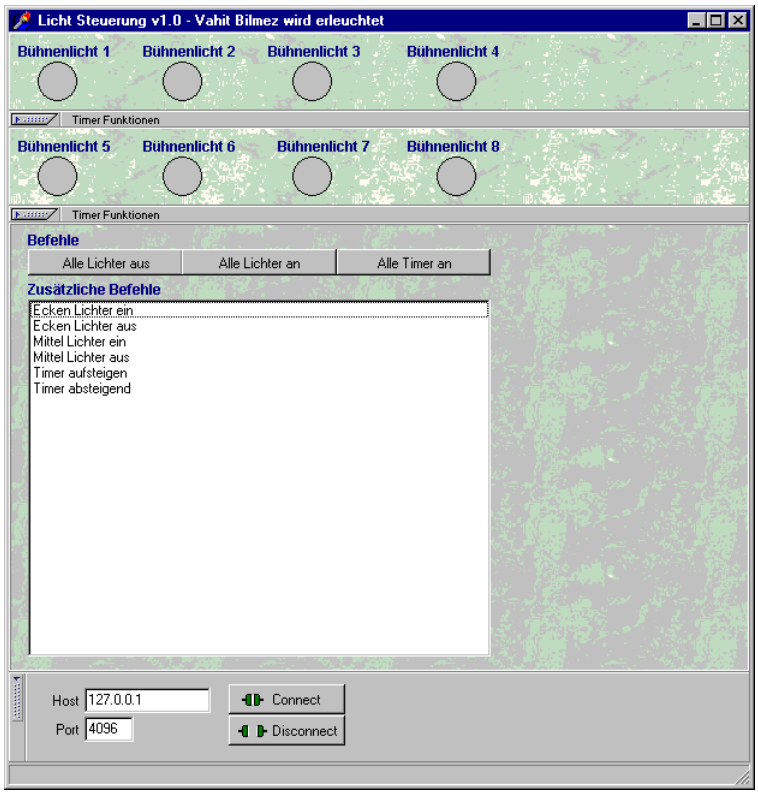

Abbildung 5.5: Die Oberflaeche ist dargestellt

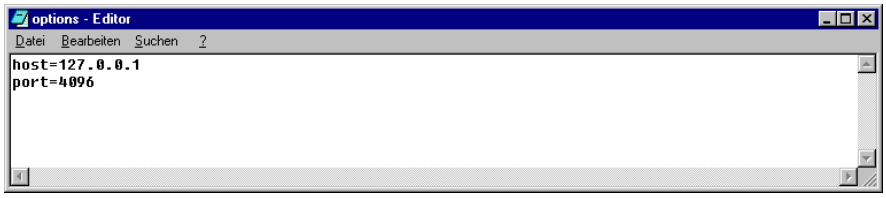

Abbildung 5.6: Durch Angabe des Hosts und des Ports kann man sich problemlos mit dem Server verbinden

zuzugreifen, der unter UNIX läuft.

5.5.5 Systemspezifikation

Motivation

Die softwareergonomischen Regeln müssen beachtet werden. Die GUI muss einfach zu verstehen sein, aber dennoch die gewünschte Funktionalität zur Verfügung stellen. Der Code muss kompakt und schnell auszuführen sein, damit die Relaiskarte über das Netzwerk schnell gesteuert werden kann.

Auswertung der Bedingungen

Die Lichtsteuerung wird so entwickelt, dass er sich mit dem Server verbinden kann. Die Lichtsteuerung kann die Verbindung mit dem Server jederzeit wieder aufnehmen.

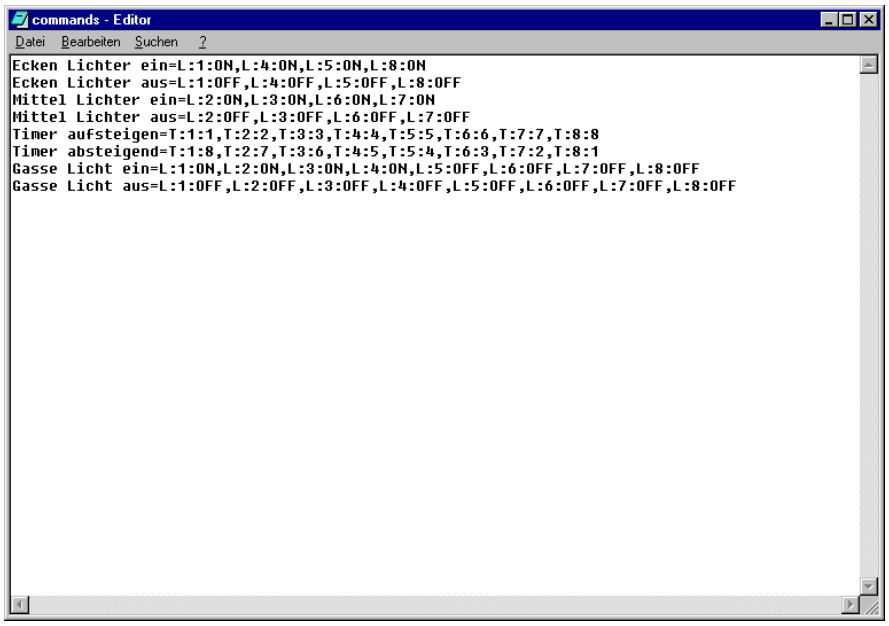

Abbildung 5.7: Editor mit dem man verschiedene Sequenzen erstellt, die anschliessend auf der GUI sichtbar werden

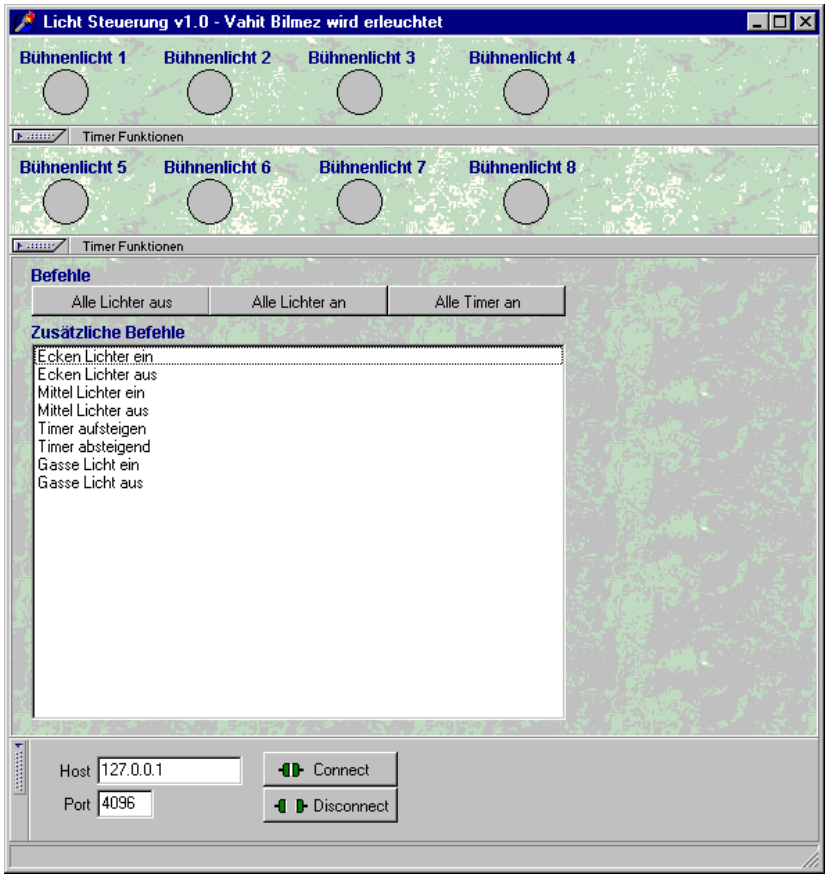

Abbildung 5.8: Die erstellten Sequenzen sind auf der GUI sichtbar

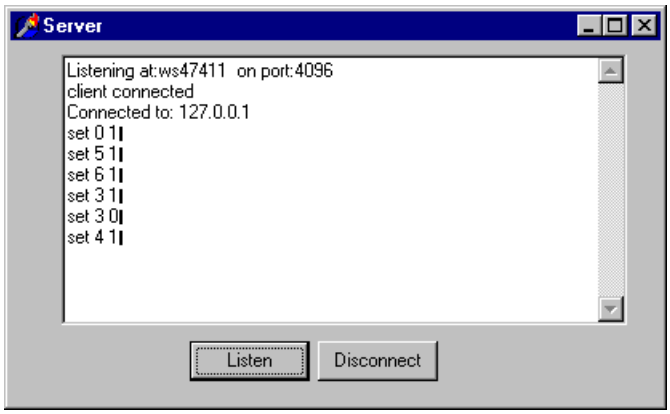

Abbildung 5.9: Protokolldatei

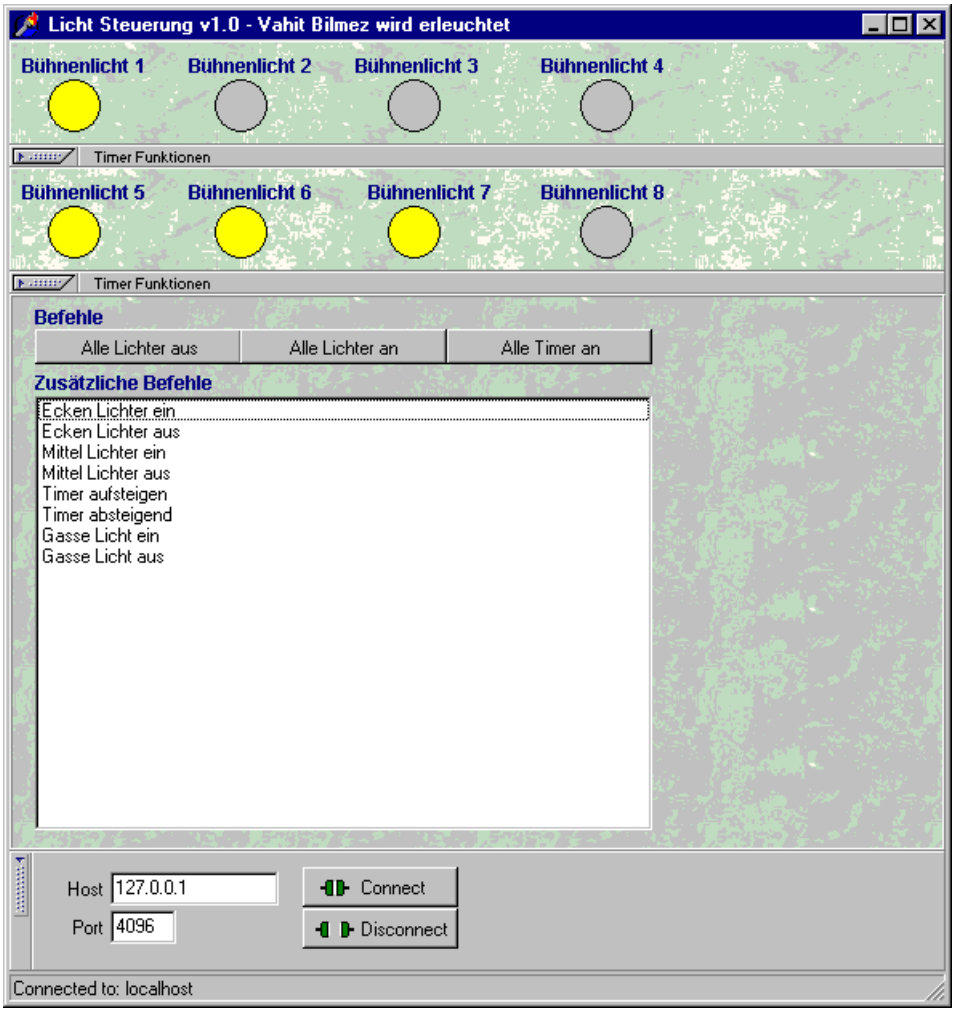

Abbildung 5.10: Auf der GUI erkennt man, dass die gelben Lichter an sind und die grauen Lichter aus. Man kann auch direkt von der Oberfläche Lichter an- und ausschalten, sowie sich mit dem Server verbinden und trennen. Die Oberfläche ist direkt manipulierbar

Kapitel 6

Fliwatüt

6.1 Allgemeiner Teil

6.1.1 Motivation

6.1.1.1 Ideenfindung (oder : Wie alles begann !)

Am Tag, als die Projekte von den einzelnen Professoren vorgestellt wurden, stellte Professor Willi Bruns sein Projekt "Theater der Maschinen" vor. Er zeigte einen Film über Chico Mac Murtrie, in dem Tiere auftauchten, die aus Altmetall zusammengebaut waren, und pneumatisch bewegt wurden. Er sagte, dass er selber nicht genau wüßte, was in diesem Projekt passieren sollte, und das es Sache der Studenten wäre dies herauszufinden.

Ein paar Wochen später trafen sich alle Leute, die das Projekt von Prof. Willi Bruns am interessantesten fanden, und meinten, etwas damit anfangen zu können. Nachdem einiges an organisatorischen Dingen besprochen war, ging es daran, festzulegen, was genau in "unserem" Projekt passieren sollte. Dies gestaltete sich schwieriger, als wir es uns alle zunächst gedacht hatten. Da wir zunächst zu keiner einhelligen Meinung kamen, beschlossen wir eine Projektfahrt zu machen, auf der wir endgültig zu einer Entscheidung kommen wollten, was genau in dem Projekt geschehen sollte.

Der "große Weg" begann auf unserer ersten Projektfahrt, auf der wir uns darüber klar werden wollten, was genau wir in unserem "Theater der Maschinen" Projekt machen wollten. Da es in einer relativ großen Gruppe (ca. 20 Personen) immer sehr schwer ist, einen gemeinsamen Konsens zu finden, teilten wir uns in Kleingruppen (ca. 4 Personen) auf. Jede dieser Kleingruppen sollte eine Idee ausarbeiten, wie sie sich das "Theater der Maschinen" vorstelle.

In unserer Kleingruppe waren wir uns sehr schnell einig, daß wir eine Maschine bauen wollten, die a) etwas erzeugen sollte und b) interaktiv mit einem Betrachter agieren sollte. Nachdem wir uns hierüber einig waren, gingen wir dazu über, diese Idee genauer auszuarbeiten.
Es kristallisierten sich zwei Ideen heraus : Das erste war eine interaktive Orgel, die je nachdem, wie sich ein Betrachter vor dieser Orgel bewegt, Töne von sich gibt. Und das zweite war ein interaktiver Mal-Roboter, der je nachdem, wie sich ein Betrachter vor einem Tisch bewegt, bestimmte Figuren zeichnet bzw. unterschiedliche Farben benutzt.

Nachdem jede Kleingruppe ihre Ideen vorgetragen hatte, diskutierten wir über die Vor- und Nachteile der einzelnen Ideen. Es wurde uns relativ schnell klar, daß wir keine einheitliche Meinung finden würden, und so entschlossen wir uns eine "Rauminstallation" zu schaffen, in der jede einzelne Idee auftauchen konnte. Dies hatte den Vorteil, daß jeder seine Ideen verwirklichen konnte.
Am Anfang hatten wir die Vorstellung eines elektronischen "Horrorkabinetts", in das der Be-

trachter hineingeht und sich dann richtig gruseln kann. Diese Idee wurde aber schon sehr bald von einer anderen abgelöst : Wir wollten eine Rauminstallation erschaffen, in die man hineingehen kann und in der einem zunächst alles ganz normal erscheint. Doch nach und nach verwandeln sich die eigentlich alltäglichen Gegenstände in Maschinen und Roboter, die dann irgendwelche Aktionen ausführen. Im Laufe des Projekts merkten wir aber recht schnell, wie schwierig es ist, Roboter als alltägliche Gegenstände zu "tarnen". Die Größe war hierbei immer das schwierigste Problem, denn wenn man einen Roboter als z.B. Coladose "tarnen" will, so muß die komplette Technik des Roboters in eben dieser Coladose verschwinden.

6.1.1.2 Gruppenbildung

Nachdem wir den groben Rahmen abgesteckt hatten, ging es in den folgenden Wochen darum, die einzelnen Ideen weiter zu verfeinern und auszuarbeiten. Und es mußten sich Gruppen bilden, die an den einzelnen Ideen mitwirken wollen.

Durch zahlreiche Diskussionen bildete sich relativ schnell eine Gruppe von vier Personen, die die Idee mit dem Roboter, der interaktiv auf Menschen reagiert, und etwas "erschafft" verwirklichen wollte. Diese Gruppe bestand zunächst aus Matthias Stojke, Matthias Liebert, Michael Triebel, und Marcus Fährer. Wir merkten relativ schnell, wie schwierig es war, selbst eine so kleine Gruppe zu koordinieren, und beschlossen deshalb die Kleingruppe nicht noch weiter zu vergrößern.
Nach ca. einem Jahr verließen Matthias Stojke und Michael Triebel aus beruflichen Gründen das Projekt und somit auch unsere Kleingruppe. Dies führte dazu, daß erheblich mehr Arbeit auf uns verbleibende Mitglieder fiel, und so einige Aspekte, die wir uns eigentlich vorgenommen hatten nicht mehr zu realisieren waren. Ansonsten war die Zusammenstellung der Gruppe sehr gut, da sich die einzelnen Interessen und das vorhandene Vorwissen sehr gut ergänzten. Wir hatten zwei Leute die sich sehr gut mit E-Technik auskannten, was die Arbeit an sehr vielen Stellen doch sehr erleichterte. Beziehungsweise wurde unsere Arbeit sehr viel zeitintensiver, nachdem eben diese beiden die Gruppe verlassen hatten. Wir waren in der glücklichen Lage zwei Leute bei uns zu haben, die ihren Facharbeiter- / Gesellenbrief hatten. Hierdurch war es uns möglich, die Werkstätten im GW-2 - Gebäude zu benutzen. Dies war gerade in der Anfangszeit sehr von Nutzen, da wir fast die kompletten Einzelteile unseres Malroboters selber gefertigt haben.

6.1.1.3 Ideenausarbeitung

Nachdem sich unsere Kleingruppe gefunden hatte und das Grobziel gesteckt war, ging es in der folgenden Zeit darum, das Unterprojekt näher zu spezifizieren und festzulegen, was genau geschafft werden sollte. Wir entschlossen uns dazu, die Idee mit dem "Mal-Roboter" weiter zu verfolgen, da dieser uns mehr Möglichkeiten bot, unsere verschiedenen Ideen weitestgehend zu verwirklichen. In zahlreichen Diskussionen legten wir fest, wie der Malroboter aussehen sollte (die Grundform) und welche Eigenschaften er haben sollte bzw. was für Aktionen er ausführen können sollte.
Bei diesen Diskussionen wurde auch der Name geboren, der uns das gesamte Projekt über erhalten bleiben sollte :

"Fliwatüt"

Das "Fliwatüt" sollte am Anfang des Projektes folgende Eigenschaften haben:

- sich autonom bewegen

- über einen Computer steuerbar sein
- Selbstlokalisation auf einem Tisch, d.h. es sollte selber "merken", wenn es in die Nähe der Tischkanten fährt und dieser dann automatisch ausweichen
- verschiedene Farben einsetzen
- die einzelnen Stifte mit sogenannten Muscle-Wires bewegen
- interaktiv auf die Zuschauer vor dem Tisch reagieren
- feste Sequenzen abfahren
- Sequenzen laden / speichern können
- Sequenzen, die das "Fliwatüt" abgefahren hat sollen gespeichert werden können
- Kommunikation mit den anderen Teilprojekten

6.1.1.4 Die ersten Schritte und die ersten Probleme (oder : Von der Idee zum fertigen Produkt)

Nachdem die theoretischen Überlegungen abgeschlossen waren, wollten wir uns voller Tatendrang dran machen, und das "Fliwatüt" bauen. Dies stellte uns aber zunächst vor die folgenden vier Probleme :

- Wie soll die Grundplatte aussehen ?
- Was für Motoren benutzen wir ?
- Wie werden die Motoren auf der Grundplatte befestigt ?
- Was für Räder benutzen wir ???

Für die Form der Grundplatte einigten wir uns auf ein Achteck. Diese Form hatte unserer Meinung nach den Vorteil, daß man die Motoren, die Sensoren und die Stiftabsenkungsvorrichtungen an einer gerade Kante anbringen kann. Dies hatte auch einen optischen Vorteil.

Bei den Motoren entschieden wir uns für Schrittmotoren, da man diese sehr gut ansteuern kann. Außerdem hatte ein Projektmitglied noch welche über, die er uns erst einmal zur Verfügung stellte. Es sollten zwei Motoren zum Einsatz kommen, anhand der das "Fliwatüt" dann in der Lage sein sollte, vorwärts, rückwärts, links vorwärts, rechts vorwärts, links rückwärts und rechts rückwärts zu fahren, sowie sich auf der Stelle links herum und rechts herum zu drehen.

Nachdem wir wußten wie die Motoren und die Grundplatte aussahen, beschlossen wir die Motoren mit einfachen Winkeln an der Grundplatte zu befestigen.

Wir benötigten zwei verschiedene Arten von Rädern. Zum einen die Antriebsräder, zum anderen die Stützräder. Bei den Antriebsrädern entschieden wir uns für Rillenreifen, mit einem Durchmesser von 81mm. Als Stützräder sollten Spornräder mit einem Durchmesser von 31mm zum Einsatz kommen. Alle Räder bekamen wir relativ preiswert bei einem ortsansässigen Elektrogroßhändler.

Da wir diese "Probleme" nun gelöst hatten, wollten wir dazu übergehen die Grundplatte zu konstruieren. Dies erwies sich zunächst schwieriger als angenommen. Das Problem, welches wir zu lösen hatten, war folgendes : Die achteckige Grundplatte sollte aus einem quadratischen

Rohling heraus geschnitten werden, der eine Seitenlänge von 30cm hatte. Da der Durchmesser 30cm ist, müssen die einzelnen Seiten des Achtecks wie lang sein ??? An diesem Problem "vergnügten" wir uns einen gesamten Vormittag, ohne zu einer brauchbaren Lösung zu kommen.

Da wir leider keine Formelsammlung dabei hatten, lieferte uns der Zufall die Lösung : Wir sahen durch Zufall eine Postkarte, auf der ein achteckiges STOP-Schild abgebildet war. Jetzt mußten wir nur noch kurz einen Strahlensatz anwenden und fertig war das Ergebnis. Der Vollständigkeitshalber haben wir an dieser Stelle noch einmal die korrekte Berechnung eines Achtecks aufgeführt :

n Eckenanzahl
α Mittelpunktswinkel
β Eckenwinkel
D Umkreisdurchmesser

$$l = D \cdot sin\left(\frac{180}{n}\right)$$

$$\alpha = \frac{360}{n}$$

$$\beta = \frac{180 \cdot (n-2)}{n}$$

Nachdem nun dieses große Problem gelöst war, konnten wir damit beginnen die Grundplatte vom "Fliwatüt" zu fertigen. Es vergingen dann doch noch einige Tage, die wir damit verbrachten herauszubekommen, wer für die Werkstätten im GW-2 zuständig ist, und die benötigten Nachweise zu erbringen, daß wir (bzw. einige von uns) die Maschinen die dort standen auch benutzen durften. Als diese Hürden genommen waren, konnte es endlich losgehen. Wir haben dann wirklich die Grundplatte an unserem ersten Tag in den GW-2 - Werkstätten gefertigt. Und da wir noch Zeit hatten, haben wir die Achsen für die beiden Antriebsräder gleich noch mit angefertigt.

GESCHAFFT !!! Der erste Schritt war getan, und wir fühlten uns gut, da wir endlich etwas hatten, was man in die Hand nehmen konnte, und etwas, das man auch mal vorzeigen konnte.

Bei unserem nächsten Besuch in den Werkstätten haben wir dann die Winkel für die Radaufhängung gekantet und die Gewindelöcher für die Madenschrauben in die Antriebswellenkupplung gebohrt.
Nachdem wir dann die Aufnahmebolzen für die Stützräder hergestellt und mit Gewinden versehen hatten, begannen wir damit, unsere Einzelteile zusammenzubauen. Bei der Montage achteten wir darauf, die beiden Antriebsräder parallel zueinander anzubringen. Dies sollte verhindern, daß das "Fliwatüt" nur zu Kurvenfahrten fähig ist. Die Parallelität wurde durch die Anbringung einer Gewindestange sichergestellt.

Als wir dann die Motoren, die Antriebsräder und die Stützräder zusammengebaut hatten, bemerkten wir, dass wir einen entscheidenden Fehler bei der Konstruktion der Stützräder gemacht hatten: Wir hatten den Drehpunkt der Stützräder genau in die Mitte gesetzt. Durch einige Versuche fanden wir sehr schnell heraus, dass dies nicht sehr sinnvoll war, da sich so die Stützräder bei einem Richtungswechsel nicht zwangsweise mit gedreht haben. Dies hatte zur Folge, dass die Stützräder blockierten. Sie waren also mehr eine Bremse als eine Stütze. Als Lösung für dieses Problem überlegten wir uns, den Drehpunkt der Stützräder aus der Mitte herauszunehmen, so wie es bei Einkaufswagen der Fall ist. Nachdem wir dies erledigt hatten,

taten die Stützräder genau das, wozu sie auch gedacht waren.

Um die Schrittmotoren ansteuern zu können, wurde eine Schrittmotorkarte besorgt. Über diese sollte die Ansteuerung vonstatten gehen. Wir brauchten aber einige Zeit, bis wir verstanden, wie die Schrittmotorkarte angesteuert wird.

In dieser Zeit fertigten wir den Prototypen eines Armes, der später einmal die Stifte absenken sollte. Nachdem wir den Arm vollendet hatten und ihn auf der Grundplatte befestigt hatten, gingen wir dazu über, einige Versuche mit den "Muscle-Wires" zu machen. Diese sollten ja die Stifte heben und absenken.
Es war nicht möglich, den direkten Weg mit den "Muscle-Wires" zu überwinden, da der Zugwinkel zu groß war. Um dieses Problem zu lösen, befestigten wir ein Rad in der Mitte des Armes, über das wir die "Muscle-Wires" laufen ließen. Nach diesen Änderungen funktionierte das Ganze sehr viel besser. Wir konnten den Arm heben und ihn wieder absenken lassen.
Trotzdem diese ersten Versuche sehr gut verliefen, beschlossen wir später, die "Muscle- Wires" doch nicht zu verwenden. Dies hatte hauptsächlich drei Gründe :

- Zur Anhebung der Stifte hätten die "Muscle-Wires" unter Dauerstrom stehen müssen. Da wir aber ja ein autonomes Gefährt konstruieren wollten, mußten wir den gesamten Stromverbrauch so gering wie möglich halten, um genügend Ressourcen für den Antrieb zu haben.
- Die "Muscle-Wires" waren relativ träge und konnten nicht den erforderlichen Druck auf die Stifte ausüben, der nötig gewesen wäre, um diese gleichmäßig auf der Oberfläche zeichnen zu lassen.
- Die "Muscle-Wires" waren nicht fehlertolerant. Dies äußerte sich z.B. darin, daß sie sehr schnell rissen, wenn der angelegte Strom auch nur minimal zu hoch war.

Wir überließen das Anheben und Absenken der Stifte im Endeffekt Servomotoren, die sich auch sehr viel leichter ansteuern ließen.

Nachdem wir die Ansteuerung der Schrittmotoren über die Schrittmotorkarte hinbekommen hatten, tauchte das Problem auf, daß bei zu kleinen Pausen zwischen den Motorschritten, d.h. bei zu hoher Geschwindigkeit, die Reifen immer durchdrehten.
Zur Lösung dieses Problems versuchten wir zunächst den Anpressdruck der Reifen zum Boden zu erhöhen, indem wir das Gewicht vom "Fliwatüt" mit einigen Gegenständen erhöhten. Dieses Vorgehen brachte aber nur minimalen Erfolg, und die Reifen drehten immer noch durch.

Unser zweiter Lösungsansatz war, die Geschwindigkeit der Motoren zu senken. Hiermit hatten wir Erfolg, was sich aber leider auch auf die Geschwindigkeit vom "Fliwatüt" auswirkte. Um die Geschwindigkeit vom "Fliwatüt" doch wieder zu erhöhen, bauten wir ein Getriebe aus zwei Zahnrädern zwischen die Motoren und die Räder. Die hierdurch erreichte Geschwindigkeit war annehmbar, und wir beschlossen es hierbei zu belassen.

Als nächstes diskutierten wir darüber, ab das "Fliwatüt" wirklich autonom werden sollte, oder nicht. Wir kamen zu dem Ergebnis, daß es auf jeden Fall wünschenswert ist, das "Fliwatüt" autonom agieren zu lassen. Nachdem wir einige Erkundigungen eingeholt hatten, bestellten wir das ATMEL STK 200 FLASH MICROCONTROLLER STARTER KIT. Im gleichen Atemzug bestellten wir ebenfalls ein paar Microchips (EDE 1204), über die die Ansteuerung der Motoren sich vereinfachen ließ.

Es dauerte einige Zeit, bis wir mit den ATMEL-Chips sehbare Ergebnisse erzielten, da wir uns zunächst mit dem ATMEL-Assembler vertraut machen mußten. Nachdem diese anfängli-

chen Schwierigkeiten überwunden waren, schafften wir es, die Motoren über die ATMEL-Chips zu beschleunigen, und zu steuern.

Nachdem wir die Stromversorgung auf Akkus umgestellt hatten, war es geschafft : Das "Fliwatüt" konnte sich autonom bewegen. Die Programmierung der zu fahrenden Strecke war aber sehr mühsam, da man alles in Assembler eingeben mußte. Da dies nicht sehr benutzerfreundlich war, beschlossen wir, ein C++-Programm unter Windows zu schreiben, welches a) eine grafische Oberfläche besitzt, b) die direkte Steuerung des "Fliwatüt" ermöglicht, c) erlaubt eine Sequenz zu zeichnen und diese direkt zum "Fliwatüt" zu schicken, d) abgefahrene Sequenzen zu speichern und e) die gespeicherten Sequenzen wieder zu laden. Dies waren die Funktionen, die zunächst geplant waren. In der Entwicklungsphase des Programms, kamen dann noch verschiedene andere Funktionen hinzu, die bei Programmbeschreibung Erwähnung finden.

Zu guter letzt bauten wir noch die "Arme" vom "Fliwatüt", die dafür zuständig waren, die Stifte abzusenken. Hierzu nahmen wir zwei Servomotoren, die jeweils einen Arm steuerten. An einem Arm waren jeweils 2 Stifte angebracht, so daß wir in der Lage waren mit vier verschiedenen Farben zu malen. Die Stifte konnten auch über unser Windowsprogramm angesteuert werden.

6.1.1.5 Die Aufführung

Für die Aufführung entschieden wir uns, das "Fliwatüt" manuell über den Computer zu steuern. Dies war notwendig, da es zu viele Eventualitäten gab, die auftreten konnten. Schon wenn die Räder nur etwas durchdrehten, fuhr das "Fliwatüt" in die falsche Richtung, und es bestand die Gefahr, daß es von der Bühne fahren könnte. Ebenso gab es bei den Schauspielern immer einen gewissen Grad an Improvisation, so daß dies auch immer von Hand ausgeglichen werden mußte. Über die manuelle Steuerung war es kein Problem, spontan auf alle Eventualitäten zu reagieren.

Natürlich war das hierfür notwendige Kabel, etwas störend, und das "Fliwatüt" wurde in seiner Beweglichkeit etwas eingeschränkt. Man mußte zu jeder Zeit aufpassen, daß das "Fliwatüt" nicht über das Kabel fährt bzw. sich nicht in ihm verheddert. Nachdem wir die Steuerung des Programms von der umständlichen Maussteuerung, auf die bedienungsfreundliche Tastatursteuerung umgestellt hatten, war es ohne Probleme möglich, sämtliche Funktionen des "Fliwatüt" mit einer Hand fernzusteuern, ohne daß es zu Verzögerungen durch lange Fahrtwege der Maus kam. Diese Steuerung hat sich bei den Theateraufführungen dann auch sehr bewährt.

6.1.2 Hardware - Überblick

- 2 Rillenreifen bzw. 2 Stollenreifen
- 2 Schrittmotoren
- 2 Standard Servos
- Atmel AVR Evalutionboard STK200
- 9V und 5V Spannungsversorgung auf dem Fliwatüt
- 20 P-130 SCR-1Z Hochstromentladezellen 1300mAh, Entladestrom: 13A
- 30V externe Spannungsversorgung via Labornetzgerät
- 2 Antriebswellen

- 2 zehner Zahnräder
- 2 vierziger Zahnräder
- 2 Winkel als Motorhalterung
- 1 FT639 zur Servoansteuerung
- 2 L293 Push-Pull 4 Channel Driver
- 2 EDE1204 Bi-Polar Stepper Motor IC
- 2 Reflexlichtschranken CNY70
- 2 Encoderscheiben (32 Segmente)
- ca. 4m RS232 Verbindungkabel
- diverse Gummibänder zur Stifthalterung
- zwei Holzstäbe als Stiftaufnahme
- 4 Eddings
- Kinderfahrradfahne für die Aufführung
- diverse Dioden, Widerstände, Platinen und Kleinkram

6.1.3 Software - Überblick

Eingesetzt wurde zur Steuerung des Fliwatüts ein Atmel AVR AT90S8515. Dies ermöglichte dem Fliwatüt:

- autonom zu agieren
- festgelegte Sequenzen abzufahren
- die Stifte zu heben und zu senken

Die PC-Software besitzt folgende Funktionalität:

- Das Fliwatüt kann in Echtzeit gesteuert werden
- Die Stifte können zu jeder Zeit gehoben und gesenkt werden
- Das Generieren von abzufahrenden Strecken, die in das Fliwatüt zu übertragen werden können
- Das generien, speichern und editieren von Sequenzen (siehe **??** Sequenzgenerierung, Seite **??**).
- Das Aufzeichnen des Streckenverlaufs bei der Steuerung in Echtzeit mittels Odometrie (siehe: 6.2.4.1, 79)

6.2 Spezieller Teil

6.2.1 Hardware

6.2.1.1 Architektur

Die Architektur des Unterprojektes Fliwatüt besteht aus einem Zusammenspiel unterschiedlichster Komponenten. Zum einem agiert das Fliwatüt gänzlich unabhängig von jeglicher Eingriff von außen. Dies gilt auch für die Stromversorgung. In dem direkten Zusammenspiel mit dem PC, der eine Echtzeitsteuerung ermöglicht, wird auf eine externe Spannungsquelle nicht verzichtet, da die Leistung der Akkus hinsichtlich Stromabgabe und Zeit begrenzt sind.
Die untenstehende Zeichnung soll den Zusammenhang zwischen den einzelnen Komponenten verdeutlichen. So gibt es gewisse "umweltliche" Einflußnahmen auf die Steuerung des Fliwatüts und "Regelkreisläufe".
Es existieren bidirektionale Schnittstellen zwischen dem PC und dem Fliwatüt und zwischen dem PC und der Corba-Umgebung (MCP). Sie dienen der jeweiligen Einlußnahme, wobei die Einflußnahme über das MCP zunächst über den PC und von dort aus zum AVR gehen. Der AVR kann direkt über seine "Senorik" beeinflußt" und diese algorithmisch verarbeiten.

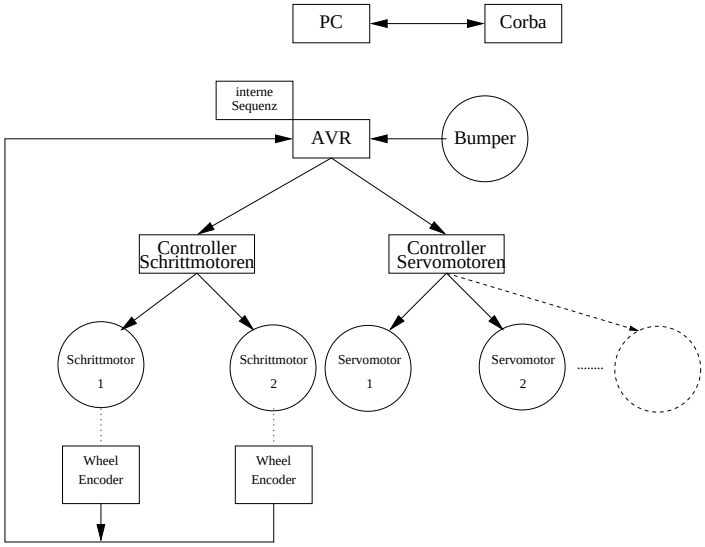

Abbildung 6.1: Beispiel

Das Gehirn des Fliwatüts ist das AVR IC. Dort werden die Motoren und Servos gesteuert und die Signale der "Sensorik" und für die Kommunikation zwischen den Teilen erzeugt / Verarbeitet. Die Abbildung 3 zeigt die einzelnen Komponenten mir ihrer Kommunikationsrichtung unabhängig von der eingesetzten Software (siehe Software AVR). Die einzelnen Kästen stelle die "Module" der Software dar. So ist die Kommunikation über die RS232 Schnittstelle aufgetrennt in eine Send und in Receive-Einheit. Beide können Interrupt gesteuert kommunizieren. So werden z.B. Steuerbefehle vom PC empfangen und über die Output-Einheit wird das Feedback an die PC Software zurückgegeben. Die 8 Bit Motorsteuerung steuert direkt das Motor-Driver IC an, während der Wheel-Encoder ein Feedback über die Drehgeschwindigkeit der Räder liefert. Die Komponenten Zeitsteuerung und Taktgenerator

6.2.1.2 Steckerbelegungen

6.2.1.2.1 Stecker Belegung vom Motor-Driver IC zum Stepper-Motor Die in der Tabellen angebenden Zahlen und Farben sind auf der Platine bzw. auf den Kabeln zu finden. Eine Fehlbelegung führt zu einem nicht funktionieren der Motoren und äußert sich durch unkon-

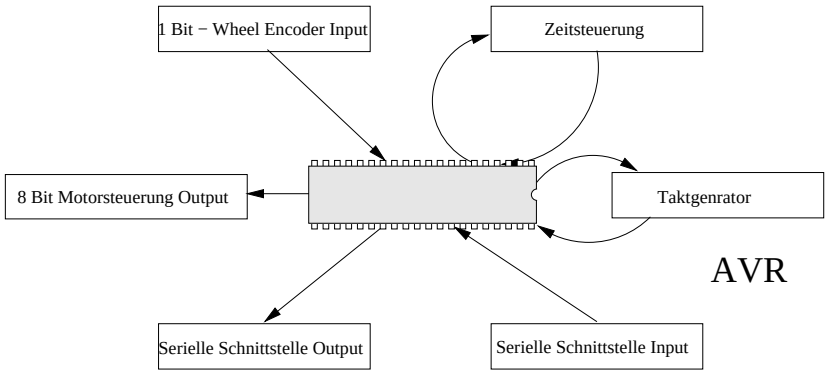

Abbildung 6.2: Schema AVR

trollierte Motorbewegung. Weiterhin ist auf eventuelle "Kurzschlüsse" zu achten, die durch die nicht abisolierten Enden der Stecker und deren geringe Abstand an der Platinenbuchse auftreten können.

Stepper-Motor	Braun	Rot	Gelb	Blau
Motor-Driver	4	3	1	2

Tabelle 6.1: Steckerbelegung Motor-Driver IC - Stepper-Motor

6.2.1.2.2 Steckerbelegung vom Motor-Driver zum Atmel AVR Die Pins Speed A, B und C geben die Geschwindigkeit an (siehe Tabelle unten). Wobei die Bitfolge 000 nicht den Stopzustand darstellt. Das Bit das zwischen Run und Stepp-Mode schaltet muß auf low gesetzt werden. Das Stepp-Mode Bit ist nicht relevant, da es nur von Bedeutung ist, wenn das Run und Stepp-Mode Bit auf high ist. Der Half/Full Steps-Mode produziert Signale im vollem oder halben Schritt, sinnvoll ist das Bit auf high zusetzen, um größere Geschwindigkeiten zu erreichen. Das Bit sieben gibt die Richtung an (Uhrzeigersinn oder nicht) in die sich die Motoren drehen sollen. Das achte Bit setzt unabhängig von den davor getätigten Einstellung die Motoren in einem Modus indem die Räder freibeweglich sind.

Nase				
	Speed A (1)	Speed C (3)	Step Mode (5)	Direction (7)
	Speed B (2)	Run/Step Mode (4)	Half/Full Steps (6)	Free Spin (8)

Tabelle 6.2: Pinbelegung Stecker Motor-Driver IC - Atmel AVR

6.2.1.3 Geschwindigkeiten bei fester Zuordnung im FullStep-Modus und im HalfStep-Modus

Bei übersetzen Getrieben müssen die Umdrehungsgeschwindigkeiten jeweils durch den Übersetzungfaktor dividiert werden, um die reale Geschwindigkeit zu ermitteln. Bei dem Fliwatüts wurde mit einem Übersetzungsverhältnis 1:10 gearbeitet, so daß sämtliche Umdrehungsgeschwindigkeiten durch den Faktor 10 zu dividieren sind. Es ist zu beachten daß eine Beschleunigung von dem Zustand 000 in den Zustand 111 nicht möglich ist. Es muß hier auf eine langsamere Beschleunigung durch Nutzung der anderen Zustände geachtet werden. Eine ausschließliche Nutzung des Zustandes 111 im FullStep-modus über einen längeren Zeitraum, ist nur bei einer Temperaturüberwachung des H-Brücken L239 IC zu empfehlen. Durch die maximale Stromaufnahme der Motoren gerät der L239 an seine thermalen Leistungsgrenzen.

Stufe	Bitfolge	RPS	RPM
1	000	0,152	9,1
2	001	0,172	10,3
3	010	0,2	12
4	011	0,244	14,6
5	100	0,303	18,2
6	101	0,4	24
7	110	0,606	36,4
8	111	1,18	70,6

Tabelle 6.3: Geschwindigkeiten im FullStep-Modus

Stufe	Bitfolge	RPS	RPM
1	000	0,077	4,6
2	001	0,089	5,3
3	010	0,103	6,2
4	011	0,121	7,27
5	100	0,154	9,2
6	101	0,2	12
7	110	0,303	18,2
8	111	0,606	36,4

Tabelle 6.4: Geschwindigkeiten im HalfStep-Modus

6.2.1.4 Bumper

Die Bumper sind vorne und hinten an der Grundplatte angebracht, um Hindernisse in der jeweiligen Fahrrichtung detektieren zu können. Pro Bumper kommen zwei Mircoschalter mit abgerundeten Schalter zum Einsatz, um auch Hindernisse, die im spitzen Winkel zum Fliwatüt stehen, festzustellen da ein "verhacken" der Schalter dadurch minimiert wird. Die beiden Microschalter sind mit einer I/O Line mit dem AVR verbunden.

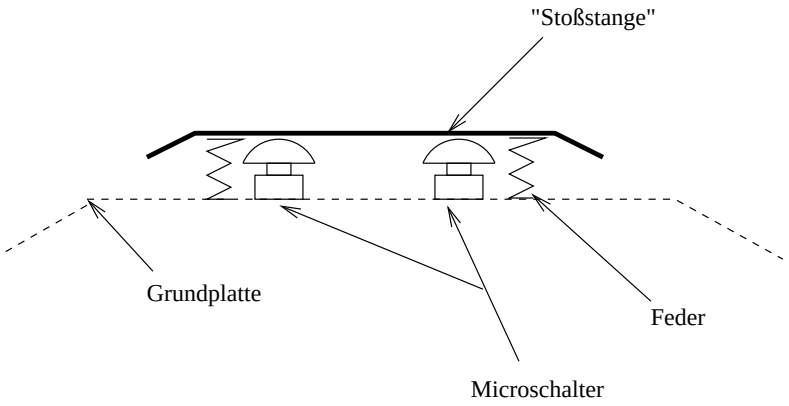

Abbildung 6.3: Schema Bumper

6.2.1.5 Servo - Stift

Die Stifte sind auf eine Art Wippe angebracht. Diese Konstruktion ermöglicht es zwei Farben pro Servo einsetzen zu können. Die Befestigung der Stifte auf der Wippe selbst ist über Gummibänder realisert, um ein "nachgeben" der Stifte zu ermöglichen und somit eine Dauerbelastung / Überlastung der Servomotoren zu verhindern. Ein weiterer Vorteil dieser Konstruktion ist es auf unebenen Flächen, wie z.B. Rauhfaserntapeten, zeichnen zu können, da fortwährend ein Anpreßdruck besteht, der die Unebenheiten ausgleicht.

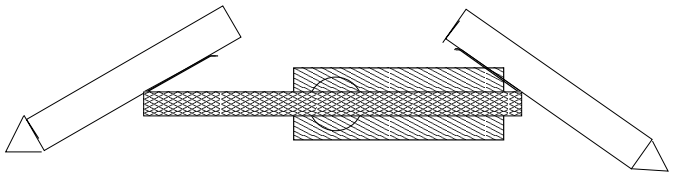

Abbildung 6.4: Stifthalterung

6.2.1.6 Schematischer Aufbau

Die Motoren des Fliwatüts sind an der Unterseite der Grundplatte befestigt. Auf den Motoren ist das Antriebszahnrad mit seinen 40zig Zähnen angebracht und treibt das kleinere Zahnrad mit den zehn Zähnen an. An dem kleinem Zahnrad ist die Antriebswelle befestigt die das jeweilige Rad antreibt. Fixiert ist die Welle durch Lagerböcke und dem Außenwinkel. Jeweils zwei Stellringe hinter dem Lagerbock verhindern ein verrutschen der Welle. Es wurde festgestellt, daß das Übersetzungsverhältnis 10:1 bestmögliche Übersetzung darstellt. Eine noch größere Übersetzung hätte den Vortrieb zu langsam gemacht und eine kleinere Übersetzung hätte die Gefahr mit sich gebracht, das die Motorleistung für einige Fahrmanöver zu gering wäre und es so zu einer Blockade gekommen wäre.

6.2.1.7 Schematischer Aufbau-Antrieb

Die untenstehende Abbildung zeigt den Aufbau des Antriebes. Das eingesetzte Getriebe entspricht einer Übersetzung von 1:10.

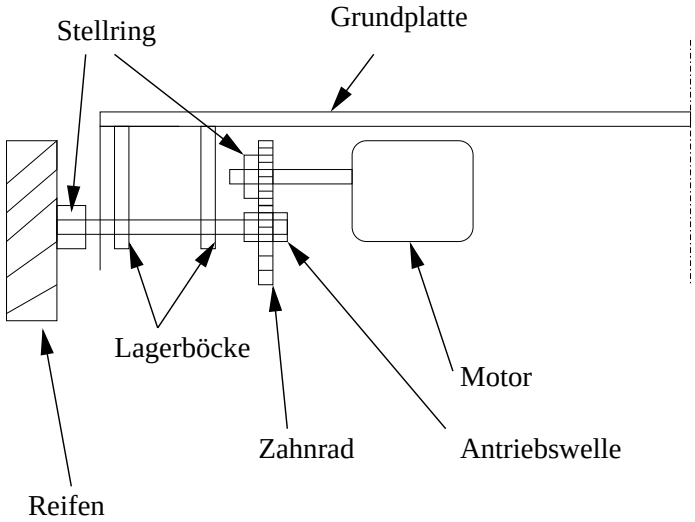

Abbildung 6.5: Antriebsmechanik Fliwatüt

6.2.1.8 Platinen

6.2.1.8.1 Motorsteuerplatine Die Motorsteuerplatine ist mit den beiden Motorcontrol IC EDE1204 bestückt, die durch den Atmel AVR direkt über die 8 I/O Lines verbundnen sind. Die von den ICs erzeugten Signale schalten zwei L293, die den Arbeitsstrom für die beiden Schrittmotoren schalten. Abgesichert ist die Schaltung mit jeweils einer 1A Diode pro Kanal gegen Spannungsrückflüsse.

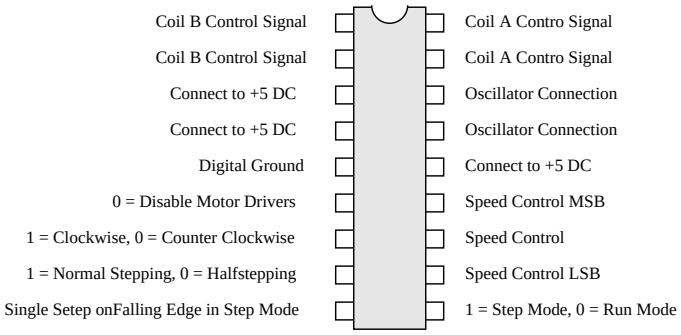

Abbildung 6.6: Schema EDE 1204 IC

6.2.1.8.2 Servosteuerplatine Die Servo-Platine ist recht einfach gehalten da nur in eine Richtung kommuniziert wird. Die untensehende Grafik gibt den schematischen Aufbau wieder. Die RS232 Receive Leitung ist mit zwei Widerständen (20 KOhm, 10 KOhm) bestückt, um die übliche 12V Normspannung der RS232 Schnittstelle für die Servo IC konforme Spannung von 5V zu reduzieren. Weiterhin wird das IC mit Masse und der 5V Betriebsspannung versorgt. Die Verbindung zu den Servo werden über einen drei Pinstecker pro Servo hergestellt. Die Spannungsversorgung des IC und der Servo ist getrennt, aufgrund der anfänglichen hohen Stromaufnahme der Servo und um ein reseten des IC zu verhindern.

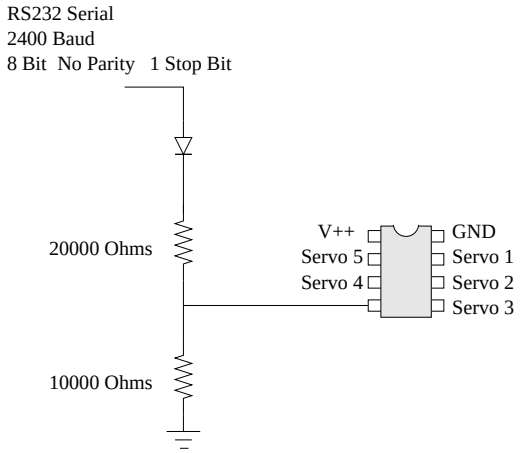

Abbildung 6.7: Aufbau Servo-Platine

6.2.1.8.3 Encoderplatine Die Reflexlichtschranke CNY70 wird über einen 15 kOhm Widerstand direkt an die I/O Leitungen des AVRs angeschlossen. Jedes mal wenn sich die schwarz/-weißen Segmente der Encoder an der Reflexlichtschranke, angetrieben durch die Drehbewegung der Räder, vorbeibewegen wird ein Signal erzeugt. Dieses Signal dient der Auswertung der Drehbewegung der Räder und soll die Odometrieberechnnung, basierend auf Geschwindigkeit und Zeitdauer der Bewegung, unterstützen. Da das generierte Signal des CNY70 für die Auswertung zu schwach ist, um durch den AVR registriert zu werden wurde ein Vorwiderstand verwendet. Die Encoderscheiben sind direkt auf der Innenseite der Räder aufgebracht und können verschieden viele Segmente enthalten. Um so mehr Segmente die Encoderscheibe besitzt, um so differenzierter kann die Bewegung registriert werden. Die höhere Differenzierung der Bewegung bringt aber auch den Nachteil mit sich, daß die Positioniergenauigkeit von Encoderscheibe und Reflexlichtschranke steigt. Es hat sich in der Praxis erwiesen, daß eine Encoderscheibe von 32 Segmenten für die Feststellung der Bewegungen in Punkte Genauigkeit und Differenzierbarkeit ausreichend ist. Das Korrigieren eines eventueller Schlupf der Räder z.B. bei der Anfahrt, ist nur durch einen Algorithmus möglich, der die bekannten Leistungswerte (in diesem Fall des Motor in Verbindung mit der gewählten Signalisierung) mit den tatsächlichen erzeugten Umdrehungen vergleicht. So ist es möglich diese für die Odometrie wichtigen Werte zu einem gewissen Grad zu korrigieren. (siehe 6.2.4.1 Odometrieberechnung, Seite 79)

6.2.1.9 L293

Die H-Bridge L293 ist eine Leistungsstufe, die mittels TTL Signalen gesteuert wird. Ihre Aufgabe ist es höhere Ströme zu schalten als sie durch die Motorsteuer IC mit ihrer TTL Spezifikation möglich sind. So können mit L293 Spannung im Bereich von 4,5V bis 36V geschaltet werden, bei einer Stromlast von 600mA bis 1A. Bei den höheren Leistungen ist aber dringend anzuraten die L293 ICs mittels eines passiven Kühlkörpers zu kühlen.

6.2.1.10 IC-Motor

Bei dem EDE 1204 Bi-Polar Stepper Motor IC handelt es sich um ein 18 Pin IC. Es ist konzipiert einen Bi-Polaren Motor zu steuern.(siehe auch Schrittmotoren) Dabei ist es möglich die Taktung von dem Chip oder auch von externe Quelle generieren zu lassen. Die dabei erzeugten Signale sind TTL Kompatibel und werden in unserem Fall durch eine L293 Dual H-Bridge weiterverarbeitet. Die einzelnen Parameter des IC werden direkt durch die I/O Lines des AVRs angesprochen.

Folgende Werte lassen sich beeinflussen:

- Drehrichtung
- Half- or Fullstepping
- Geschwindigkeit (siehe Tabelle Geschwindigkeiten)
- Stepmode
- Freespin

6.2.1.11 IC-Servo

Die Ansteuerung der Servos erfolgt über einen FT639 Servo Controller Chip der Firma FerretTronics. Der FT639 ist ein Chip zur Ansteuerung von RC Servos. Es können bis 5 Servo von diesem Chip gesteuert werden. Angesprochen wird der Chip über die RS232 Schnittstelle des STK200 . Der Chip verarbeitet die Kommandos die er vom AVR IC bekommt und generiert daraus die Steuersignale (Pulsweitenmodulation) für die Servos (siehe Tabelle unten). Der Chip unterscheidet zwei veschiedene Betriebsarten:

6.2.1.11.1 Initialisierung In diesem Modus werden noch keine Positionsdaten übertragen. Hier wird die

- Startposition der Servos definiert, in die sie nach dem Wechsel in die aktive Betriebsart fahren
- Ursprung
- Pulslänge Hier kann zwischen kurzen und langen Pulsen gewählt werden. Damit kann festgelegt werden, ob der Servo im 90 Grad oder im 180 Grad Bereich arbeiten soll.

Kommando	Binär	Erläuterung
Ursprungswert	0110xxxx	siehe folgende Tabelle
kurzer Puls	01010101	
langer Puls	01011010	
Positionskommandos	0sssllll 1sssuuuuu	siehe tabelle Positionskommandos
Umschalten auf aktive Betriebsart	100	

Tabelle 6.5: Kommandos in der Setup-Betriebsart

```
0sssllll  1. Byte für die Servoposition
1sssuuuu  2. Byte für die Servoposition
llll die unteren 4 Bits des Positionsbyte
uuuu die oberen 4 Bits des Positionsbytes
```

```
sss Servonummer: 000 = Servo 1 / 001 = Servo 2 / 010 = Servo 3
011 = Servo 4 / 100 = Servo 5
xxxx  Ursprungswert
```

Ursprungswert	kurzer Puls	langer Puls	Kontrollbyte
0	.147 ms	.237 ms	0110 0000
1	.219 ms	.357 ms	0110 0001
2	.291 ms	.477 ms	0110 0010
3	.363 ms	.597 ms	0110 0011
4	.435 ms	.717 ms	0110 0100
5	.507 ms	.837 ms	0100 0101
6	.579 ms	.957 ms	0110 0110
7	.651 ms	1.077 ms	0110 0111
8	.723 ms	1.197 ms	0110 1000
9	.795 ms	1.317 ms	0110 1001
10	.867 ms	1.437 ms	0110 1010
11	.939 ms	1.557 ms	0110 1011
12	1.011 ms	1.677 ms	0110 1100
13	1.083 ms	1.797 ms	0110 1101
14	1.155 ms	1.917 ms	0110 1110
15	1.227 ms	2.037 ms	0110 1111

Tabelle 6.6: Urpsrungswerte in der Setup-Betriebsart

6.2.1.11.2 Aktive Betriebsart In der aktiven Betriebsart werden die Steuerimpulse für die Servos erzeugt. Damit die Servos "eingeschaltet". Es gibt nur zwei Kommandos welche in der aktiven Betriebsart gesendet werden können: Positionskommandos (siehe untenstehende Tabelle) und das Kommando zur Rückkehr zur Setup Betriebsart.
Mit Hilfe der Positionskommandos kann das Servo bewegt werden. Die neue Position wird angefahren, sobald das zweite Positionsbyte gesendet wurde. Es werden die gleichen Kommandos benutzt wie in der Setup Betriebsart.

6.2.1.11.3 Positionskommandos Um eine neue Position an ein bestimmtes Servo zu senden, müssen zwei Bytes gesandt werden. Das erste byte enthält die unteren 4 Bit (lower nibble) das zweite Byte die oberen 4 Bit (upper nibble) der Servoposition. Das Byte mit dm unteren Bits muß vor dem Byte mit den oberen Bits gesendet werden. Siehe auch unten stehende Tabelle:

6.2.1.12 Die Schrittmotoren

Die von uns eingesetzten Schrittmotoren sind bipolaren Typs. Bipolare Motoren sind fast identisch wie unipolare Motoren aufgebaut mit dem unterschied, daß die Wicklungen einfacher konstruiert sind, indem sie keine center taps besitzen. Die Motoren sind zwar einfacher aufgebaut, aber die Steuerschaltung muß sicher stellen, das sie die Polarität für jede Wicklung umkehrt. So ist ein Einsatz einer H-Bridge (siehe 6.2.1.9,71) unbedingt erforderlich, denn sie gewährleistet, das jede Wicklung unabhängig, die Polarität geändert werden kann. Untenstehend ist die Kontrollsequenz aufgezeigt, wobei + und - die Polarität die Spannungsversorgung der jeweiligen Wicklung angeben.

```
                Fullstep          Halfstep
Wicklung 1a +---+---+---+---  ++--++--++--++--
```

Servonr.	Pos. Dezimal	Pos. high/low	1. Kommando	2. Kommando
1	0	0000 0000	0000 0000	1000 0000
1	49	0011 0001	0000 0001	1000 0011
1	185	1011 1001	0000 1001	1000 1011
1	255	1111 1111	0000 1111	1000 1111
2	0	0000 0000	0001 0000	1001 0000
2	49	0011 0001	0001 0001	1001 0011
2	185	1011 1001	0001 1001	1001 1011
2	255	1111 1111	0001 1111	1001 1111
3	0	0000 0000	0010 0000	1010 0000
3	49	0011 0001	0010 0001	1010 0011
3	185	1011 1001	0010 1001	1010 1011
3	255	1111 1111	0010 1111	1010 1111
4	0	0000 0000	0011 0000	1011 0000
4	49	0011 0001	0011 0001	1011 0011
4	185	1011 1001	0011 1001	1011 1011
4	255	1111 1111	0011 1111	1011 1111
5	0	0000 0000	0100 0000	1100 0000
5	49	0011 0001	0100 0001	1100 0011
5	185	1011 1001	0100 1001	1100 1011
5	255	1111 1111	0100 1111	1100 1111

Tabelle 6.7: Kommandos in der Aktiven-Betriebsart

```
Wicklung 1b --+---+---+---+-  --++--++--++--++
Wicklung 2a -+---+---+---+--  -++--++--++--++-
Wicklung 2b ---+---+---+---+  +--++--++--++--+
```

Anzumerken ist, das der Betrieb im FullStep-Modus zwar höhere Geschwindigkeiten ermöglicht, aber daß das Drehmoment im HalfStep-Modus wesentlich höher ist. Zu erklären ist an dem Zusammenspiel der Wicklungen. Während im FullStep-Modus nur eine Wicklung zur Zeit aktiviert ist, sind im HalfStep-Modus immer zwei Wicklung gleichzeitig aktiviert. Dies ist auch aus dem Diagramm oben zu entnehmen. Die entwickelte Kraft des Motors ist also immer ein Kompromiss zwischen Geschwindigkeit und Kraft. In unserem Unterprojekt haben wir das Fliwatüt überwiegend im FullStep-Modus betrieben, da das von den Motoren entwickelte Drehmoment immer für die Erfordernisse ausreichend war.

6.2.2 Softwareentwicklung AVR

6.2.2.1 Software AVR

Es gibt zwei verschiedene Softwareversionen für den Betrieb der AVR ICs.

6.2.2.1.1 Assembler - Version Die Softwareversion die ganz in Assembler geschrieben wurde bietet folgende Features:

- Sequenzablaufsteuerung mit freiwählbarer Zeitsteuerung
- Taktgenerierung für die Schrittmotoren mit Beschleunigung
- Auswertung und Verarbeitung der "Sensorsignale"

Die Sequenzabfolge wird "festcodiert" und kann während der Laufzeit nicht mehr beeinflußt werden mit der Ausnahme bei einem Ansprechen der Bumpersensorik. Für die Erstellung der Sequenzabfolge können die einzelnen Befehle für die Motorsteuerung über die Driver IC direkt als 1 Byte kodiert werden oder auf die komfortableren Makros zurückgriffen werden, die meisten Befehle zu Verfügung stellen. Zwischen den einzelnen Befehlen muß eine Zeitverzögerung definiert werden, auch hier kann man zwischen "festen" Makrowerten oder der direkten Zeiteingabe wählen. Da die eigentlichen Verzögerungswerte spezifisch bedingt, durch die schnelle Rechenleistung der RISC Prozessoren und der nur 8 und 16 Bit großen Timer, nur sehr kurze Verzögerungen zu lassen, sie befinden sich üblich im Bereich von Millisekunden, mußten die Timer kaskadiert werden (siehe 6.2.2.2 Timerkaskadierung, Seite 75).

Des weiteren kann die Software auch die Signale für die Schrittmotoren generiert. Dabei werden die I/O Leitungen direkt an die Leistungsstufe L293 angeschlossen (pro Motor 4).

6.2.2.1.2 C - Version Die C Softwareversion bietet folgende Features:

- Sequenzablaufsteuerung mit freiwählbarer Zeitsteuerung
- Auswertung und Verarbeitung der "Sensorsignale"
- Interrupt gesteuerte Kommunikation über RS232 Schnittstelle zum PC
- "Steuerung" der Servos via RS232-Schnitstelle
- Verarbeitung der Steuerbefehle vom PC für die Realtimesteuerung

Der große Unterschied zu der Assembler Version, ist die Integration der Kommunikation über RS232 in die Software. Erst dies ermöglicht eine Echtzeitsteuerung durch den PC (siehe Beschreibung PC-Software). Die Kommunikation ist auf der AVR Seite so gelöst, das durch den Einsatz einer interruptgesteuerten Lösung, die Rechenleistung der anderen Programmteile nur sehr gering beeinflußt wird. So ist es möglich neben einer "festen" Sequenzabfolge jederzeit spontan in die Abfolge einzugreifen, um z.B. die weitere Ablaufsteuerung manuell zu betreiben. Es können nicht nur Abläufe der Antriebsmotoren in die Sequenz integriert werden, sondern es ist ebenfalls möglich die Servosteuerung für die Stiftabsenkung mit aufgenommen werden. Die Steuerung der Servomotoren läuft ebenfalls über die interrupt gesteuerte Kommunikation via RS232 ab. Dabei werden die Steuersignale an das Servosteuer IC geschickt, der die Befehle verarbeitet und dann die Steuersignale an die jeweiligen Servomotoren schickt (siehe 6.2.1.11 IC-Servo, Seite 72).

6.2.2.2 Timerkaskadierung

Hintergrund Wenn man bei einer Mikrocontrolleranwendung lange Verzögerungszeiten benötigt stehen einem mehrere Möglichkeiten zur Realisierung zur Verfügung.
Eine Möglichkeit ist es den internen Timer des Mikrocontroller zu benutzen. Da diese Timer meistens zwischen 8 und 24 Bit groß sind, erreicht man bei einem 24 Bit Timer mit einer Prozessortaktfrequenz von 1 MHz gerade Verzögerungen von wenigen zehntel Sekunden. Diese Alternative ist nur wirklich eine, wenn der Prozessortakt entsprechend gering ist.
Eine weitere Lösung wäre per Software die Timer-Overlows zu zählen, die entstehen, wenn der Timer sich resetet. Die Lösung verkompliziert die Softwareentwicklung nicht unerheblich und birgt Probleme in der Behandlung, wenn der Prozessor in den Sleep-Modus geht und auf einen Timer-Overflow wartet, der somit nicht mehr stattfinden kann (Deadlock). Eine dritte mögliche Lösung wäre einen langsamen Oszillator zu verwenden. Der Timer wird somit zum externen Eventcounter. So lassen sich extrem lange Verzögerungen erreichen. Hier ist aber wieder der Nachteil, daß "Hardware" implementiert werden muß.
Wir haben uns für eine Timerkaskadierung entschieden, die im Falle eines Sleep-Mode funktioniert.

Der AVR 8515 besitzt zwei Timer, einen 8 und einen 16 Bit Timer. Beide Timer besitzen eine prescaling Einheit und können als interne Clock oder als Counter verwendet werden, die einen externen Ausgang besitzen um das Counting zu triggern.

6.2.2.2.1 8 Bit Timer/Counter Der 8 Bit Timer kann als Source CK, prescaled CK oder über einen externen Eingang wählen. Die relevanten Register sind Timer/Counter0 Control Register (TCCR0), das overflow status flag als TIFR und die Interrupt enable/disable Settings in der Timer/Counter Interrupt Mask Register (TIMSK).

6.2.2.2.2 16 Bit Timer/Counter Der 16 Bit Counter kann ebenfalls die Source als CK, prescaled CK oder über einen externen Eingang wählen. Die wesentlichen Funktionen, wie overflow, compare match, capture event und stop Settings, sind in den Timer/Counter1 Control Registern TCCR1A und TCCR1B zu finden. Die Interrupt Settings sind Time/Counter Interrupt Mask Register (TIMSK) untergebracht. Der Timer/Counter1 bietet zwei Vergleichsfunktionenregister, Output Compare Register 1A und 1B (OCR1A und OCR1B), die mit dem aktuellen Inhalten des Timer/Counter1 verglichen wird. Ebenso wird die Funktion unterstützt bei einem erfolgreichen Vergleich den Counter zu löschen und ein externes Signal zu erzeugen.

6.2.2.2.3 Hardware "Manipulation" Um die langen Verzögerungen zu erreichen werden die beiden Timer des AVR IC kaskadiert (siehe Grafik unten). Der Timer/Counter1 wird als Timer konfiguriert mit einem festen Ratio der System-Clock. Bei jedem Erreichen des festgelegten Wertes im Compare Register (OCR1AH / OCR1AL) wird ein Signal am OC1A Pin ausgelöst und das Timer/Counter Register TCNT1H / TCNT1L wird auf 0000 zurück gesetzt und startet wieder das Zählen. Das bei einem erfolgreichen Vergleich ausgehende Signal vom OC1A wird vom Pin T0 registriert und löst einen Event im Timer/Counter0 aus. Der Timer/Counter0 wird als Counter benutzt und zählt die eingehenden Signale von den Vergleichen vom Timer1. Bei einem vorher definierten Overflow wird ein Interrupt getriggert, der das Ende der Verzögerung signalisiert.

Die längste Verzögerung ist wie folgt zu berechnen:

$$T = 2IF_s * T1P + OCR1A + (256 - TCNT0)$$

wobei:
$F_s = SystemClock\ Frequency$
$T1P = prescale\ ratio\ TCCR1B$ (8, 64, 25, 1024)

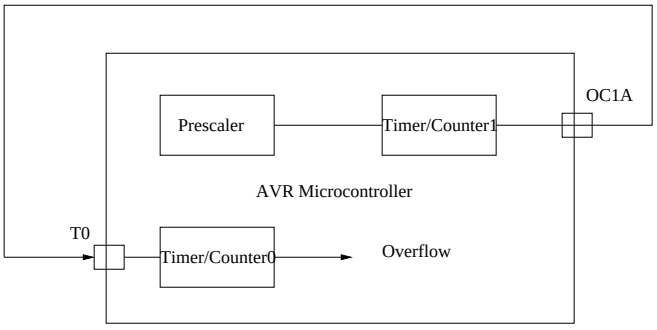

Abbildung 6.8: Kaskadierung der Timer

6.2.2.2.4 Hardware "Software Beispiele in Assembler und C"

```
        ldi     r16, $40        ; TCCR1A Signal bei jedem
        out     TCCR1A, r16     ; OC1A compare match
        ldi     r16, $01 ;<----- the output compare ...
        out     OCR1AH, r16     ; High
        ldi     r16, $20 ;<----- Low
        out     OCR1AL, r16
        ldi     r16, $20        ; Configure sleep mode
        out     MCUCR, r16
        ldi     r16, $02        ; Enable T/C0 interrupt
        out     TIMSK, r16
        ldi     r16, $80        ; Global interrupt enable
        out     SREG, r16
```

Im obigen Code-Beispiel wird gezeigt, wie die Timer/Counter präpariert werden, um bei jedem compare ein Signal auszugeben. Die mit den Pfeilen gekennzeichneten Stellen geben letztendlich die Verzögerung an, indem sie die Werte für das high / low compare Register setzen.

```
loop:
        ldi     r16, $06        ; Reload counter 0 for a division ...
        out     TCNT0, r16      ; ... ratio of 250
        ldi     r16, $06        ; Start counter 0 for ...
        out     TCCR0, r16      ; ... external pin T0 source
        ldi     r16, $00        ; Reset timer 1 value
        out     TCNT1H, r16
        ldi     r16, $00
        out     TCNT1L, r16
        ldi     R16, $0C        ; Start timer 1 for a prescale ...
        out     TCCR1B, r16     ; ... ratio of 256
        sleep                   ; Wait for delay
        ldi     r16, $00        ; Stop timer 1
        out     TCCR1B, r16
        ldi     r16, $00        ; Stop timer 0
        out     TCCR0, r16
        ret
```

In diesem Code-Beispiel wird der Timer/Counter0 als counter initialisiert und festgelegt, daß das trigger Signal auf T0 erwartet wird. Danach wird der Prozessor in den sleep-Zustand versetzt, bis das angegebene Ratio erreicht wird. Danach werden beide Timer/Counter gestoppt. Und die Abarbeitung weiteren Codes ist möglich.

Analog hierzu die Entwicklung in C für Atmel AVR:

```
CCR1A = 0x40;           // OC1A compare match

OCR1AH = 0x00;          //<----- output compare ...
OCR1AL = 0xf9;
TIMSK = 0x02;           //Enable T/C0 interrupt

DDRB = 0xff;            //PortB Output
PORTB = 0xff;
```

```
    MCUCR = 0x20;          /* Configure Sleep-Mode*/
    _SEI();                /* Global Interupt enable */
```

und das Starten und das Warten auf die Verzögerung:

```
void Delay (void)
{
        TCNT0 = 0x06;      //... ratio of 250
        TCCR0 = 0x06;      // ... external pin T0 source
        TCNT1H = 0x00;     //Reset timer 1 value
        TCNT1L = 0x00;
                           // Start timer 1 for a prescale ...
        TCCR1B = 12;       // 13/1024, 12/256, 11/64, 10/8, 9/ck
        PORTB = 0xff;
        _SLEEP();          // Wait for delay

        TCCR1B = 0x00;     //Stop timer 1
        TCCR0 = 0x00;      //Stop timer 0
}
```

6.2.2.3 Beispiel für feste Sequenzen

In diesem Abschnitt soll nur kurz skizziert werden, wie Festsequenzen sowohl in Assembler und C eingegeben werden. An dem Format könnte sicherlich noch gefeilt werden, es soll nur den aktuellen Stand der Entwicklung widerspiegeln.

Unter C:

```
_SLEEP();
PORTC = 0xf1;
_SLEEP();
PORTA = 0xe2;
```

Dabei geben die Werte die auf PortB angelegt werden die in dem oben angeführten Kapitel erläuterten Werte wieder, allerdings in diesem Fall als HEX-Wert. Die _SLEEP Aufrufe zwischen den Werten leiten die Verzögerung ein. Es können auch Werte an die _SLEEP Funktion übergeben werden, um eine zeitliche Steuerung der Verzögerungen zu erreichen.

```
        ldi r16, 0b10000111
        out PORTC, r16
        ldi r16, 0b10000111
        out PORTA, r16

        rcall   tix

        rcall   loop

        rcall  ti2

        ldi r16, 0b01100101
        out PORTA, r16
        out PORTC, r16
```

Hier wird demonstriert, wie die Sequenzgenerierung in Assembler erfolgt. Auch hier wird über die Ports A und C die Werte an den Motor-Driver übertragen. Es werden den Aufrufen auch die verschiedenen Methoden der Verzögerungseinstellungen gezeigt. Es können über Makros feste Verzögerungswerte oder per Parameterübergabe individuelle Werte eingestellt werden.

6.2.3 Software PC

6.2.4 Software-Entwicklung PC

Auf die Beschreibung der Software PC-Funktion wird nur im geringen Maße eingegangen, da die Programme recht gut kommentiert sind, erschließt sich daraus ihre Bedeutung und Funktion.

6.2.4.1 Odometrieberechnung

Für die Berechnung der Fahrwege bedienten wir uns der Odometrie. Das Verfahren beruht darauf, daß die Achsbreite, die zurückgelegte Strecke einer Radumdrehung und die Achsbreite bekannt sind. So lassen sich ausgehend von einem bekannten Startwinkel (in der Startposition 0) dem Endwinkel und die dazwischen zurückgelegte Strecke berechnen. Dabei werden die Informationen über die Zustandsänderung per Sensorik oder über die Zeit mitgeteilt.

So lassen sich die Positionsänderungen für jeden Impuls wie folgt berechnen:

Geradeausfahrt:

$$icoo_x = cos(angle_{old}) * (elapsed - time * distancePerSecond)$$
$$icoo_y = sin(angle_{old}) * (elapsed - time * distancePerSecond)$$

Rechts:

$$angle_new = (elapsed - time * angle)$$

Links:

$$angle_new = (elapsed - time * angle)$$

Zurück:

$$icoo_x = -(cos(angle_{old}) * (elapsed - time * distancePerSecond))$$
$$icoo_y = -(sin(angle_{old}) * (elapsed - time * distancePerSecond))$$

Auf der Stelle rechts:

$$angle_{new} = 2 * -angle$$

Auf der Stelle links:

$$angle_{new} = 2 * angle$$

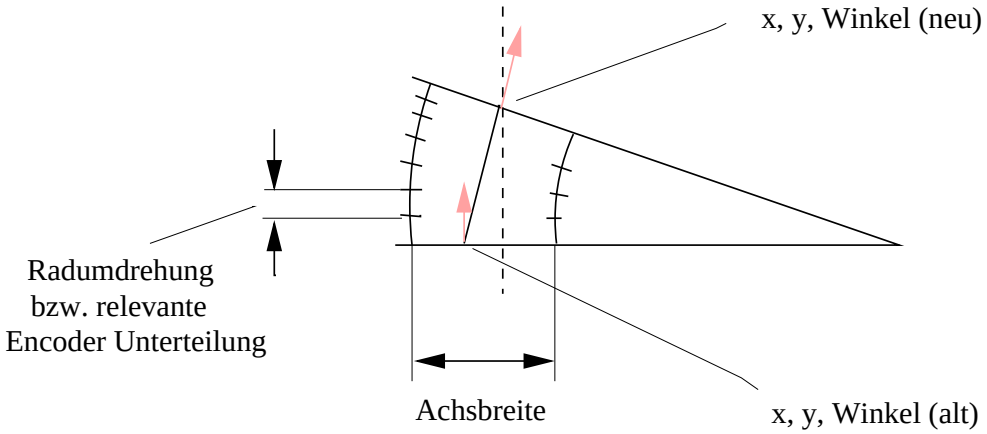

Abbildung 6.9: Odometrie-Berechnung

Es werden dabei 6 Fälle wie oben in den Formeln dargelegt unterschieden. Sie reichen vollkommen aus, die aktuelle Position zu bestimmen. Diese errechneten Werte werden in den nächsten Berechnungen wieder als Grundlage herangezogen ($angle_{old}$ usw.), um die nächsten Positionen zu berechnen. Dieses Berechnungssystem kann aber leider keine Fehler kompensieren. So wird ein eventueller Schlupf der Räder als ständige Positionsänderung interpretiert und geht somit voll in die Positionsberechnungen ein. Abhilfe kann hier nur eine Kalibrierung von Zeit zu Zeit schaffen.

6.2.4.2 Bedienung Software

6.2.4.2.1 Realtimesteuerung Die Richtungssteuerung des Fliwatüts in Echtzeit erfolgt über die **alphanumerische** Tastatur. Die Tasten sind wie folgt belegt:

- Taste 8 - Vorwärts
- Taste 2 - Rückwärts
- Taste 4 - Links
- Taste 6 - Rechts
- Taste 5 - Stop
- Taste 7 - Linkskurve vorwärts
- Taste 9 - Rechtskurve vorwärts
- Taste 1 - Linkskurve rückwärts
- Taste 3 - Rechtskurve rückwärts
- Taste - - Langsamer
- Taste + - Schneller

Das Absenken und Heben der Stifte über die Servomotoren wird ebenfalls über die **normale** Tastatur wie folgt gesteuert:

- Taste 1 - Servo 1 Farbe 1
- Taste 2 - Neutral

- Taste 3 - Servo 1 Farbe 2
- Taste 4 - Servo 2 Farbe 1
- Taste 5 - Neutral
- Taste 6 - Servo 2 Farbe 2

6.3 Probleme bei der Entwicklung

6.3.0.3 E-Technik

Durch den Abgang unserer beiden "E-Techniker" standen wir vor einem nicht unerheblichen Problem. Die Realisierung der elektrotechnischen Seite bekam nun Priorität, da wir beide noch verbliebenen Teilnehmer nicht gerade die Experten auf diesem Gebiet sind. Hierzu waren zwar schon Kleinigkeiten der Ansteuerung, die wir später allerdings wieder verwerfen mußten, realisiert, aber im Vorfeld hatten wir zuerst mechanische Probleme gelöst. Parallel galt es ebenfalls die Software (weiter) zu entwickeln.

6.3.0.4 Bodenhaftung

Ein großes Problem war die Bodenhaftung des Fliwatüts. Auf flauschigen Teppichboden fuhr das Fliwatüt recht anständig. Jedoch für die Aufführung und die universelle Einsetzbarkeit des Fliwtüts galt es sicher zu stellen, daß es auch auf anderen Untergründen gut zu recht kommt. Erste Abhilfe schufen wir in dem wir von den Rillenreifen auf Stollenreifen umstiegen. Diese boten noch immer nicht die absolute Zuverlässigkeit auf jeglichen Untergründen. Erst das Verkürzen der beiden "Stützräder" um ein paar Millimeter erhöhte den Druck auf die Antriebsräder so erheblich, das nun ein nahezu einwandfreies Fahren gewährleistet war.

6.4 Anhang

Diese Fotos sollen einige technische Eigenschaften des Fliwatüts veranschaulichen. Die Bilder entstanden während der Entwicklungsphase und den anschließenden Proben des Theaterstückes im Senatssaal des MZHs.

Die beiden Bilder 6.14 und 6.15 zeigen die Verkabelung der Schrittmotoren mit den Steuerungs-ICs und die Anbindung an den Mikroprozessor. Auf dem Foto 6.17 ist der, unter der Grundplatte angebrachte, Servomotor mit der montierten "Stifthalterung" zu erkennen. Die Abbildung 6.18 veranschaulicht die Gesamtkonstruktion des Fliwatüt.

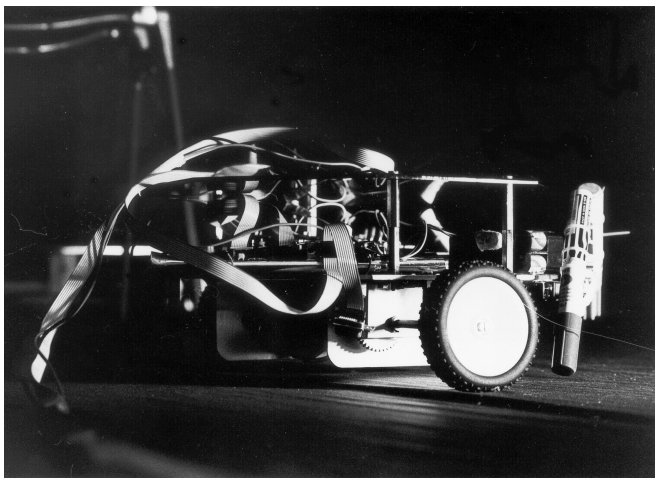

Abbildung 6.10: Fliwatüt bei den Proben - Seitenansicht

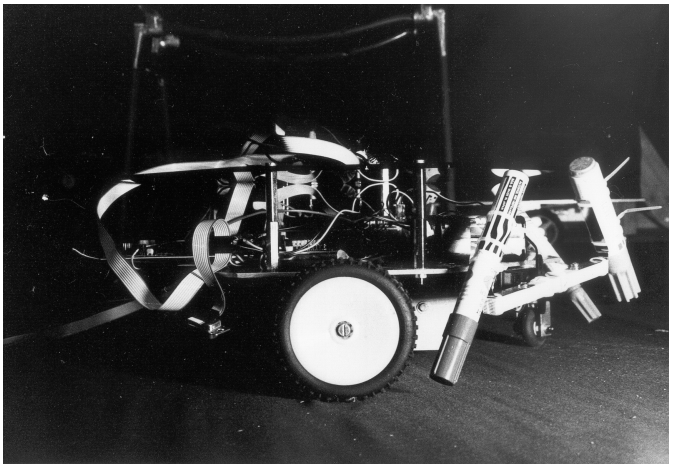

Abbildung 6.11: Fliwatüt bei den Proben - Seitenansicht der Stifthalterung

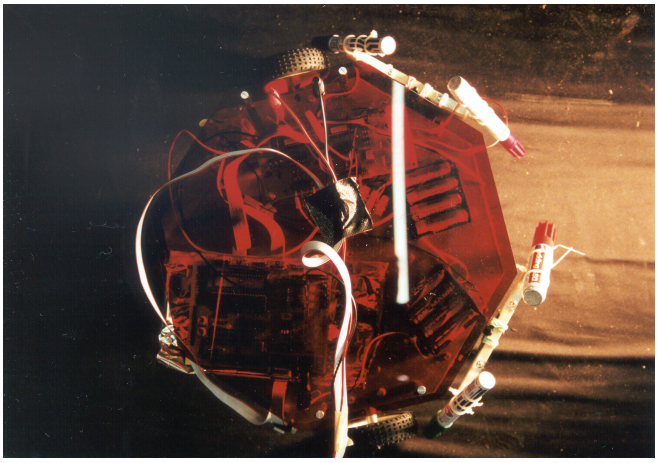

Abbildung 6.12: Draufsicht

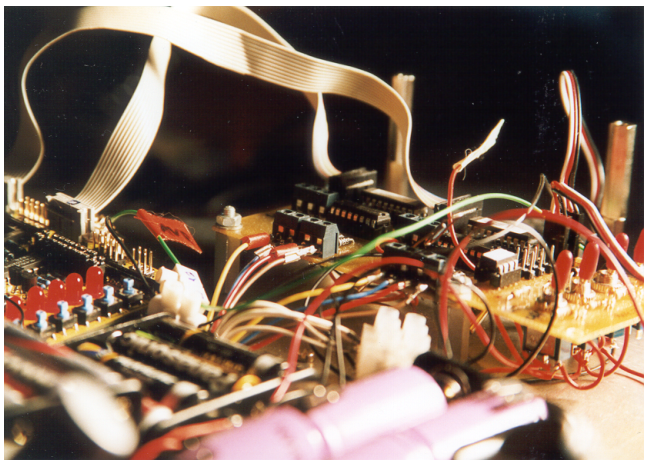

Abbildung 6.13: Aufbau und Verkabelung der Elektronik

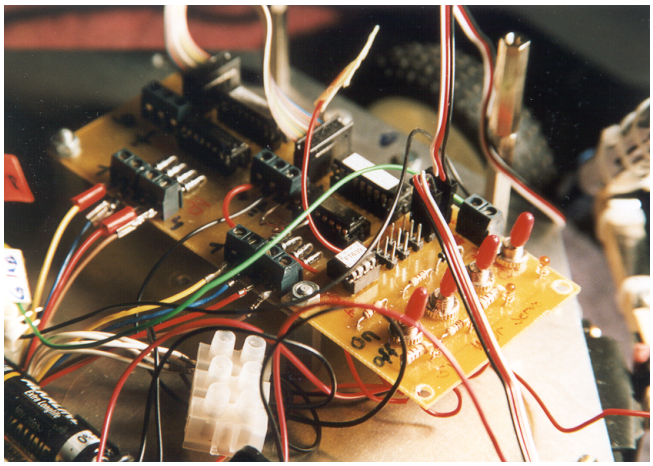

Abbildung 6.14: Detailansicht: Verkabelung der Schrittmotorsteuerung

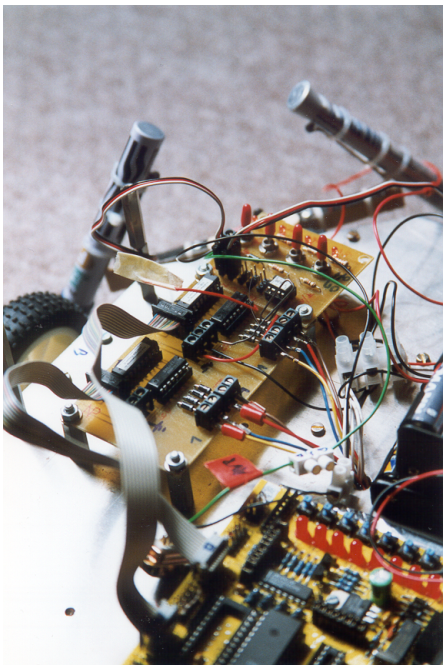

Abbildung 6.15: Detailansicht: Aufbau der Schritt- und Servomotorsteuerung

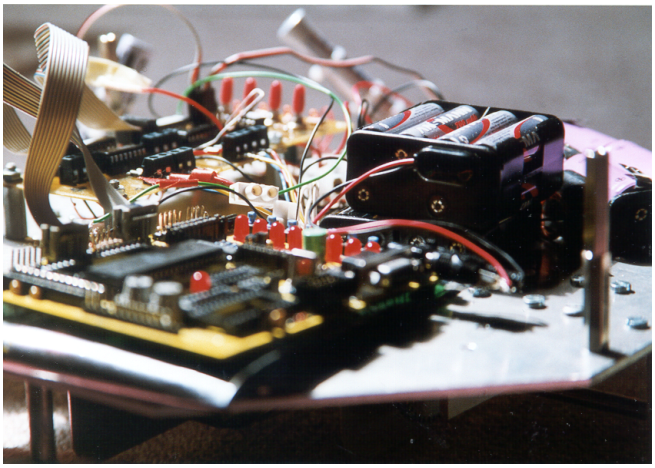

Abbildung 6.16: Spannungsquelle der Steuerung und AVR Prozessor

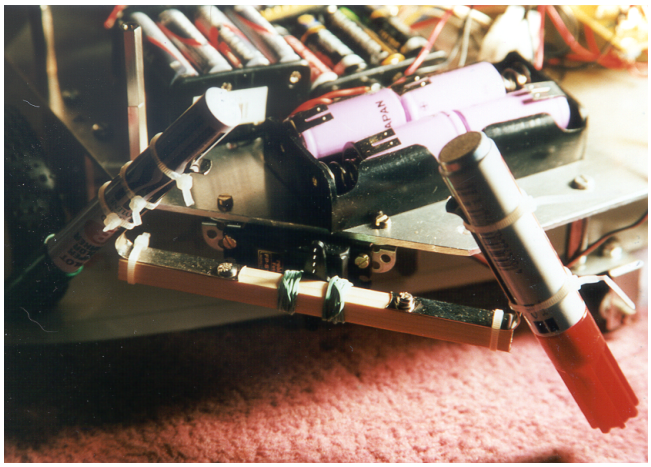

Abbildung 6.17: Detailansicht: Stiftaufhängung

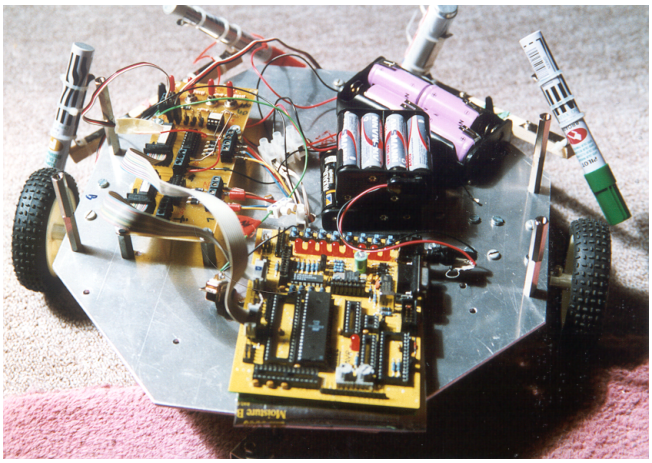

Abbildung 6.18: Draufsicht

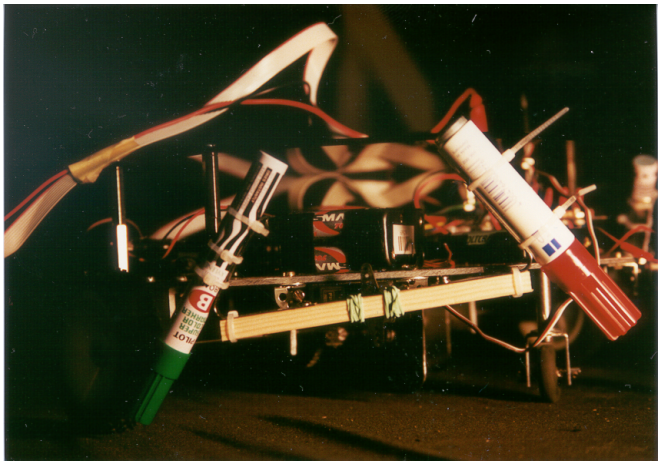

Abbildung 6.19: Fliwatüt während der Fahrt auf der Bühne

Kapitel 7

Marion

7.1 Einleitung

„Theater der Maschinen“ begann im Sommer 1998. Feste Vorgaben zum Inhalt gab es im Vergleich zu anderen Projekten nicht. Es sollte in den kommenden zwei Jahren versucht werden, ein informatisches Projekt zu erarbeiten, das anschliessend als Theater-/Bühnendarstellung präsentiert werden kann. Die Projektteilnehmer beschlossen, die ersten sechs Monate als Zeitraum für die Ideenfindung zu nutzen (dieser Prozess wird im Hauptteil dieses Projektberichtes genauer beschrieben).

Auf Anregung von Prof. Wilhelm Bruns entstand im Dezember 1998 die Idee, eine mechanisch gesteuerte Marionette und eine von Menschen gesteuerte Eingabesensorik zu entwickeln, um damit die Bewegungen eines Menschen auf die Marionette zu übertragen. Es bildete sich eine 4 köpfige Projektgruppe, bestehend aus Denis Böhme, Hartje Bruns, Jörn Raffel und Tim Evers.

Hartje Bruns und Tim Evers verliessen die Arbeitsgruppe nach ca. 18 Monaten. Im Verlauf des Projektes bildete sich eine weitere Gruppe- bestehend aus Khaled Abdelrahimsai-Pjau-, die zusätzlich zu der Marionette und der Eingabesensorik eine grafische Darstellung der Marionette auf einem PC implementierte.

7.2 Hardware

Es standen dem Projekt nur Computer auf x86-Basis zur Verfügung:

- Zwei vom Fachbereich 3 gestellte Projektrecher: AMD K6-200MHz, 32 MByte-RAM, 4.3/10GByte-Festplatte mit 17-Zoll-Monitoren. Wegen eines Fehlers in den verwendeten Prozessoren war ein Speicherausbau auf über 32MByte nicht möglich.
- Einige 386er mit 25/40MHz, 8MByte-RAM, 150-200MByte-Festplatte, 12-Zoll/15-Roll-Monitore
- Ein 486-DX4 mit 16MByte-Arbeitsspeicher und 200MByte-Festplatte; eine Leihgabe eines Projektteilnehmers.
- Ein Pentium III-600MHz, mit 3D-Voodoo-Garfikkarte und 20GByte-Festplatte. Dieser Rechner stand aber erst gegen Ende des Projekts zur Verfügung und gehört zum Bestand des Artecs.

Wir beschlossen aufgrund der vorhandenen Hardware, ein verteiltes System zu implementieren, d.h. jede Komponente (Marionette, Sensorik und die grafische Ausgabe) sollte auf einem Rechner laufen, und die einzelnen Komponenten sollten über TCP/IP kommunizieren.

Die bescheidene Rechenausstattung erwies sich als ausreichend für die Verwendung unter Linux.

7.3 Betriebssystem

Bei der Wahl des Betriebesystems sollten folgende Anforderungen erfüllt werden:

- preemptives Multitasking
- einfaches Einbinden von selbst geschriebenen Low-Level Treibern für z.B. ISA Karten
- keine Lizenzgebühren
- stabile und offene TCP/IP Implementierung

Wir entschieden uns für RedHat Linux, da alle Anforderungen erfüllt wurden und man es im nachhinein mit einem entsprechenden Kernel um Echtzeitfähigkeiten erweitern kann.

Alternativ wäre die Verwendung des kommerziellen QNX-RTOS, einer BSD-Variante oder Microsoft-Windows möglich gewesen.

QNX erschien uns als ein sehr zuverlässiges, mit allen nötigen Features versehenes Betriebssystem, welches jedoch nicht so offen ist wie Linux oder die BSD Varianten. Zudem sind Linux und die BSD-Varianten sind grundsätzlich als Quellcode verfügbar.

Bei BSD hat man die Wahl zwischen drei verschiedenen freien Versionen: FreeBSD, NetBSD und OpenBSD.

Als Microsoft-Windows wäre nur die Version Windows 98 in Frage gekommen. Andere Versionen stellten zu hohe Hardwareanforderungen(Windows NT) oder wären technisch unzulänglich (z.B. Windows 3.11). Ein Vorteil von Windows wäre ein umfangreiches Angebot an verfügbarer Software gewesen.

Linux ist ein sehr offenes, sehr flexibles und extrem konfigurierbares Betriebssystem. Es ist in verschiedenen Echtzeit- und Embedded-Versionen erhältlich.

7.4 Marion - Die mechanische Marionette

7.4.1 Übersicht

Marion ist eine vom Computer gesteuerte Marionette und besteht aus verschiedenen Holzteilen, die miteinander verbunden sind. An den Arm- und Beingelenken und am Rumpf sind Ösen angebracht, die über Fäden mit Hebeln verbunden sind. Diese werden von sieben Servomotoren angesteuert, die wiederrum an einer Servokarte angeschlossen sind. Diese ist mit dem seriellen Port eines PCs verbunden.

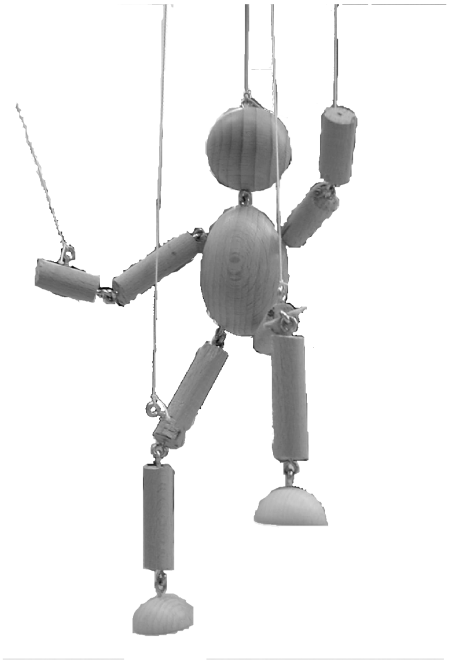

Abbildung 7.1: Marion - Der Körper

7.4.2 Mechanik

Die Mechanik wird über sieben Modellbauservos gesteuert. Jeweils ein Servo steuert die Höhe eines Arms, die Höhe eines Beins, die Drehung beider Beine, die Drehung beider Arme und die Beugung des Rumpfs. Die Servos steuern mit ihren Ruderhörnchen nicht direkt die Marionette, sondern werden durch eine Hebelkonstruktion über-/bzw. untersetzt. Die Servos lenken dabei über Stahl-Lenkstangen, die aus Aluminium gefertigte Hebel an, an denen die Fäden der Marionette befestigt wurden. Die Hebelverhältnisse können durch eine grosse Anzahl von Bohrungen in den Hörnchen und Hebeln stark variiert werden.

Bei dieser Konstruktion spielten zwei Überlegungen eine Rolle:

- Die Lager der Servos sollten nicht die Gewichtsbelastung der Marionette aufnehmen.
- Durch die Hebelkonstruktion ist eine mehrstufige Übersetzung geschaffen worden, was sich beim Experimentieren als sehr vorteilhaft rausstellte.

Die Hebel und Servos für die Auf- und Abbewegung der Arme und Beine, sind an Trägern befestigt, welchen einen Drehpunkt in ihrem vertikalen Mittelpunkt haben. Die Vereinfachung,

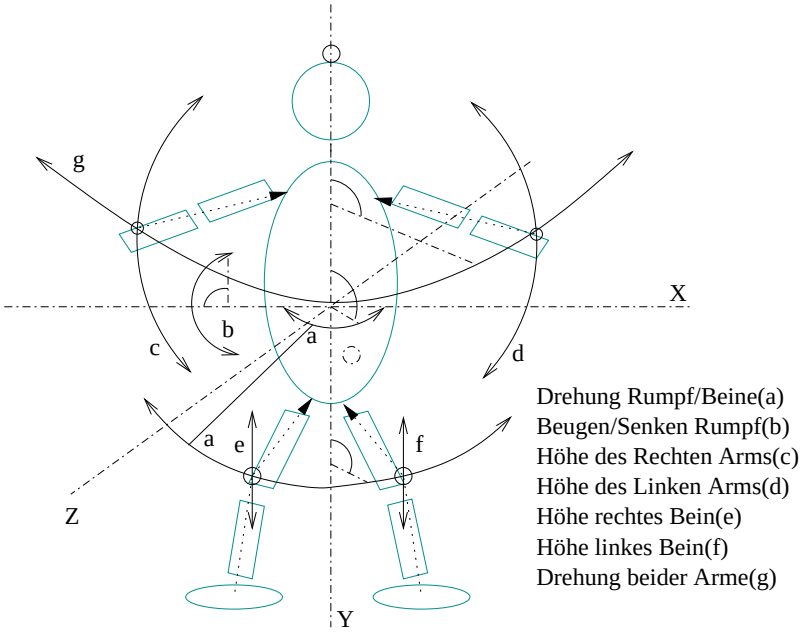

Abbildung 7.2: Marion - Die Freiheitsgrade und ihre Abhängigkeiten

das die Drehnung beider Arme bzw. Beine gemeinsam erfolgt, bedeutet, das jeweils der andere Arm bzw. das andere Bein die entgegengesetzte Position einnimmt.

Anfangs war nur die Drehung der Arme vorgesehen. Es hat sich jedoch bald rausgestellt, das bei den theoretischen Modell zur mechanischen Marionette ein wichtiger Aspekt übersehen wurde: Die Hand des Marionettenspielers am Kreuz der Marionette. Diese bietet drei weitere Freiheitsgrade, die den Bewegungsfluss wesentlich flüssiger gestalten.

Beim Spielen einer Marionette hat auch die Trägheit der Marionette eine nicht unbedeutende Rolle. Um unser Modell zu verbessern haben wir beschlossen, einen weiteren Drehpunkt für die Beine einzuführen. Dies sollte die Bewegung flüssiger gestalten.

Die Fäden, die die Beine anheben und absenken, sind nahezu senkrecht über den Beinen umgelenkt. Diese Umlenkpunkte drehen sich mit den Rumpf mit.

Zusätzlich, aber leider nicht zum Einsatz gekommen, ist eine Mechanik die Marions Drehung um die eigene Achse bewerkstelligen sollte. Letztere wird mit einem Schrittmotor angesteuert. Ein zweistufiges Getriebe soll gewährleisten, das ein ausreichendes Drehmoment vorhanden ist.

Die mechanische Konstruktion ist zum überwiegenden Teil aus Aluminium gefertigt. Im Wesentlichen aus Alu-Vierkant- und L-Profilen, die per Hand zurecht gesägt wurden.

Die Verbindung zwischen der Marionette und der mechanischen Ansteuerung wird über Fäden hergestellt, die in ihrer Länge durch Lüsterklemmen variierbar sind.

7.4.3 Elektronik

Die Ansteuerung der Servomotoren wird über eine :dbs:Karte realisiert, welche am seriellen Port des Computers angeschlossen wird. Diese Karte erzeugt die Impulse, deren Länge die Position der Servos bestimmen. An der Servokarte können bis zu acht Servos angeschlossen werden.

Die Stromversorgung der Servos bereitete Probleme, da die Stromaufnahme der Servos die Leistungsfähigkeit des Netzteils überstieg und diesen zu einen Spannungszusammenbruch des Netz-

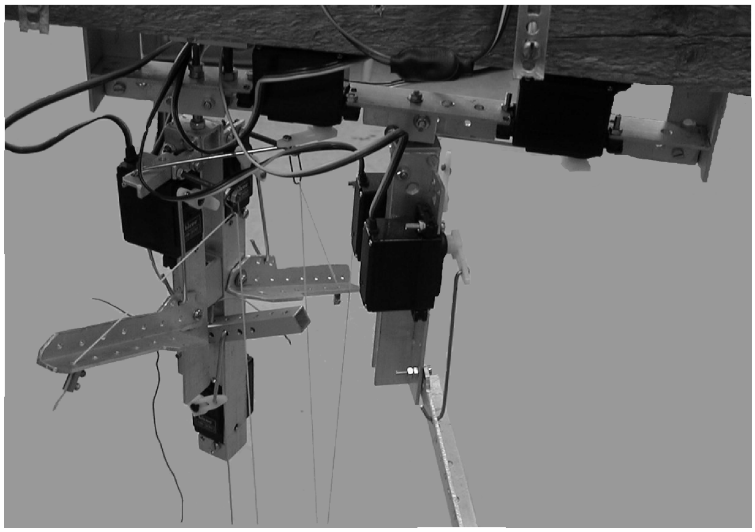

Abbildung 7.3: Marion - Die Mechanik

teils führte, sobald mehr als 4 Servos gleichzeitig im Betrieb waren. Der Zusammenbruch der Versorgungsspannung hatte zur Folge, das der Mikrocontroller auf der Servokarte abstürzte und die Marionette nur noch apathisch zuckte. Dieses Problem wurde gelöst indem die Stromversorgung auf ein 6V-Bleiakkumulator umgestellt wurde.

7.4.4 Diskussion

Alternativ wäre die Ansteuerung der Servomotoren auch über eine Real-Time-Version von Linux realisierbar gewesen. Da die serielle Schrittmotorkarte jedoch schon angeschafft wurde, wurde diese zur Ansteuerung eingesetzt. Diese Anwendung wird als klassisches Applikations-Beispiel für RTLinux verwendet :dbs:link.

Es stellte sich im Laufe der Zeit als ungünstig heraus, das sich die Spezifikationen an den Möglichkeiten der mechanischen Marionette orientierten. Es wäre besser gewesen, die Spezifikation weiter zu fassen, so das die jeweiligen Marionettenmodelle nur eine Teilmenge dieser Spezifikationen ausfüllen. Dieses Problem wurde deutlich, als die virtuelle Marionette mehr Freiheitsgrade erlaubte, als ursprünglich vorgesehen war. Allerdings wurde Anfangs nur von der mechanischen Marionette ausgegangen.

7.5 Der Galgen - Positionsbestimmung im Raum

7.5.1 Übersicht

Der Galgen erfasst die Position von Armen und Beinen eines Menschen. Er besteht aus einer Holzrahmen, an dem eine Alukonstruktion mit 4 Drucksensoren und 2 Potentiometern angebracht ist.

Die Alukonstruktion besitzt jeweils eine Schiene für die Arme und Beine. Diese Schienen sind in der Mitte gelagert, so das eine Drehung in der Horizontalen möglich ist. Die Drehung wird mit Hilfe der Potentiometer erfasst. An den Auslegern der Schienen befinden sich Drucksensoren, über die elastische Bänder laufen, die wiederum fest mit der Schiene verbunden sind. Wenn nun ein Proband unter der Alukonstruktion steht und die elastischen Bänder an Armen und Füssen befestigt, dann wird bei vertikalen Bewegungen das jeweilige Band entsprechend mehr oder

Abbildung 7.4: Marion - Die Mechanik im Detail

weniger gespannt. Dadurch wirkt auf den Drucksensoren mehr oder weniger Kraft bzw. es fliesst entsprechend der Belastung mehr oder weniger Strom. Die so abgenommenen Ströme werden mit Spannungsteilern in Spannungen umgewandelt. Diese können dann von einen AD-Wandler in digitale Werte übersetzt und Programmen zur Verfügung gestellt werden.

7.5.2 Mechanik

7.5.2.1 Gerüst

Das Gerüst besteht aus mehreren Vierkanthölzern mit 10 cm Kantenlänge, die mit Holzschrauben untereinander verschraubt sind. Nachdem wir bei dem Entwurf einen statischen Fehler gemacht hatten, wurde das Gerüst auf eine Bodenplatte aufgeschraubt, um den Galgen mehr Stabilität zu geben.

Der Galgen ist in zwei Teile zerlegbar, damit er transportiert werden kann.

Optisch gab er ein markantes Erscheinungsbild auf der Bühne.

7.5.2.2 Sensorik

Anfangs haben wir versucht, die Sensoren an einer Holzkonstruktion zu befestigen, aber es zeigte sich schnell, das Holz als Werkstoff nicht geeignet war. Wir entschlossen uns, als Werkstoff Aluminium zu benutzen. Das Resultat war weitaus besser, die Konstruktion hatte mehr Stabilität und die beweglichen Teile weniger Spiel.

7.5.3 Elektronik

7.5.3.1 Sensoren

Unsere erste Überlegung war ein System bestehend aus handelsüblichen PC-Mäusen: dem Probanden sollten an den Gliedmaßen Bänder angelegt werden, die über jeweils eine Rolle laufen.

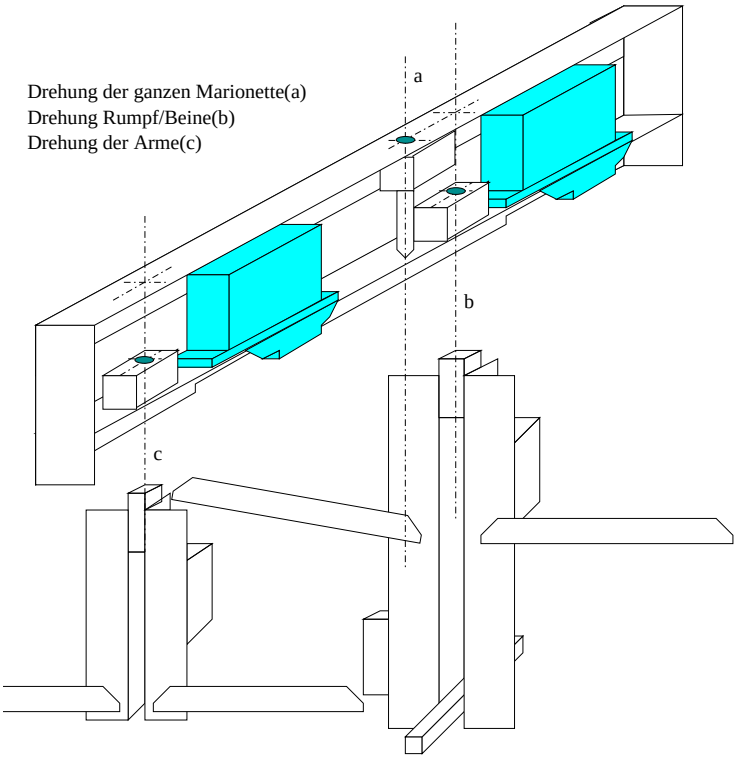

Abbildung 7.5: Die Aufnahme der Arm-, Bein- und Rumpfmechanik. Sie beinhaltet die Servos für die Drehung von Rumpf/Beine und Arme.

Die Drehungen der Rolle würden dann durch eine Lichtschranke erfasst und in elektrische Signale umgewandelt. Allerdings hat diese Methode einige Nachteile:

- Die Mechanik ist aufwendig
- Es ist anhand eines einzelnen Wertes nicht möglich die Postion zu bestimmen, da es sich um Incremental-Geber handelt
- Bei ungünstigen Lichtbedingungen kann das System versagen, bedingt durch Umgebungslicht

Schliesslich entschieden wir uns für Drucksensoren. Dem Probanden werden Gummibänder an Händen und Füssen angelegt, die über einen Drucksensor geführt werden. Je nach ausgeübtem Druck der Gummibänder verändern die Drucksensoren ihren Widerstand.

Da bei jedem Start des Systems, die Werte der Drucksensoren etwas abweichen konnten, bauten wir eine Kalibrierungsfunktion ein. Diese sollte automatisch einen passenden Arbeitspunkt für die Sensoren finden. Dieser war auch abhängig von den Proportionen des Probanden.

7.5.3.2 ADDA Karte

Für die Realisierung der Eingabe benötigten wir eine AD Wandler Karte, die die vorhandenen analogen Ströme bzw. Widerstände in digitale Werte konvertierte. Unsere Wahl fiel auf eine Karte aus dem Conrad Katalog, allerdings stellte sich schnell heraus, das diese Karte nicht wirklich brauchbar war, weil sie mehrere technische Mängel aufwies und nur unzureichend dokumentiert

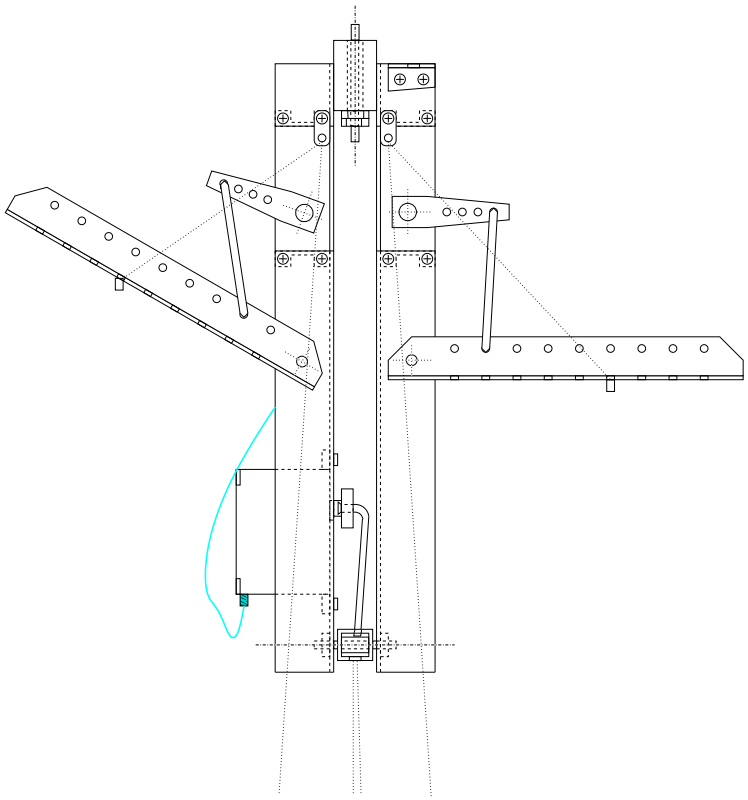

Abbildung 7.6: Die Mechanik zur Ansteuerung der Beinehöhen und der Rumpfbeugung. Die Zeichnung zeigt die Mechanik von vorne gesehen. Oberhalb der Bein-Servos befinden sich die Umlenkungen für die Bein-Fäden. Die Umlenkungen werden auf Abstand gehalten, damit sich die Fäden nicht in der Mechanik verhäddern.

war. So hatte die Karte entgegen der Produktbeschreibung nur eine Auflösung von 7 Bit anstatt der versprochenen 12, die Schalt- und Wandlungsraten waren sehr langsam und das mitgelieferte Basic Programm war nicht ansatzweise erklärt bzw. nachvollziehbar. Die Karte war unserer Meinung nach auf dem technischen Stand vom Anfang der 90er Jahre.

Nach einigem Suchen entschieden wir uns für eine ISA Karte von der Firma Kolter Elecronic. Im Gegensatz zu der vorherigen Karte traten nun keine Fehler mehr auf. Die Karte hat folgende technische Spezifikationen:

Analogeingänge Spannung:	16 single-ended, Unipolar/Bipolar
Digitalein- und ausgänge:	8TTL
Auflösung AD-Wandler:	12 Bit
Meßgenauigkeit AD-Wandler:	11 Bit
Toleranz:	$\pm$ 1/2 LSB
Meßspannungen AD-Wandler:	0...5 V unipolar, $\pm$ 2,5 Volt bipolar
Wandlerrate AD-Wandler:	typisch 25 μs
Slotbelegung:	8-Bit-Steckplatz
Externer Anschluss:	25pol. SUB-D Buchse

7.5.4 Software

Die Werte der ADDA Karte werden 10 Mal pro Sekunde ausgelesen. Dieses Intervall führte zu einer relativ flüssigen Bewegung der realen Marionette.

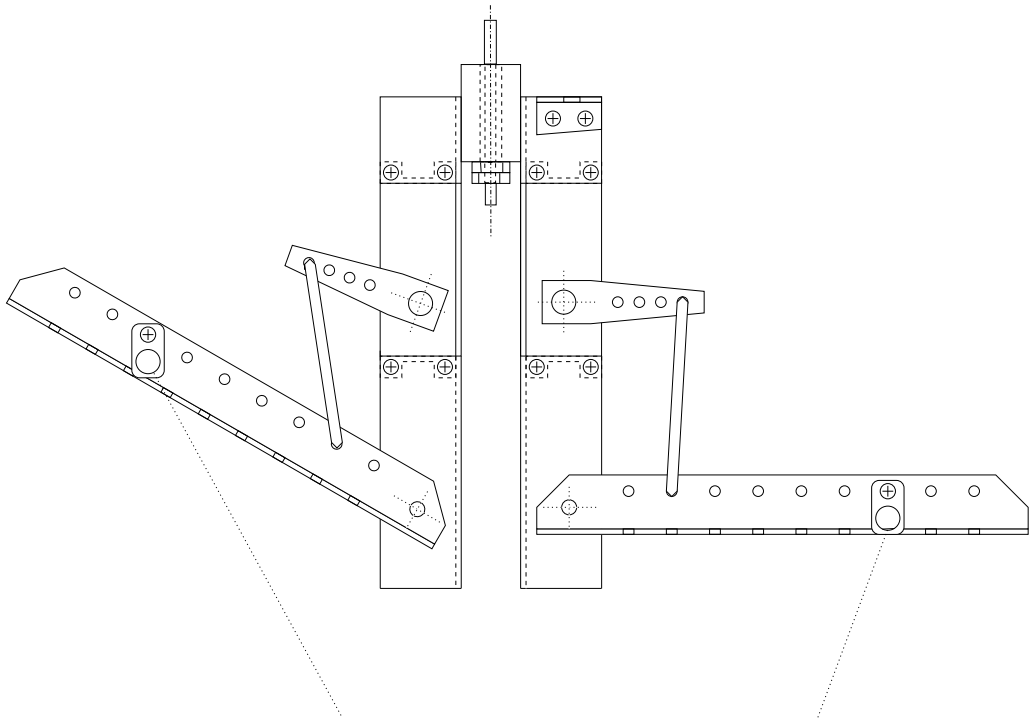

Abbildung 7.7: Die Mechanik zur Ansteuerung der Arme von vorne gesehen.

Bei der Messwerterfassung werden der maximale und minimale Wert gespeichert und bei Über- oder Unterschreitung neu gesetzt. Anhand dieser Werte wird eine Differenz gebildet und der aktuelle Messwert wird in eine Zahl zwischen 0 und 100 umgewandelt. D.h. das System justiert sich selber, es müssen keine Minimal- und Maximalwerte von Hand eingegeben werden.

Von den möglichen 4096 verschiedenen Werten der ADDA Karte werden in der Regel nur ca. 200 - 500 benutzt. Die Ursache hierfür liegt in der Spannung der Gummibänder: will man den vollen Bereich des AD Wandlers ausnutzen, wird die Spannung der Gummibänder so stark, das ein normales Bewegen des Probanden nicht mehr möglich ist.

7.5.5 Diskussion

Bei der Bewegungsverdopplung ging es um eine einfache und günstige Lösung. Für einen Betrachter der Performance, ist die gewählte Lösung eine der Eindruckvollsten.

Die Positionserfassung des Galgens ist relativ ungenau, ausserdem wird nur die vertikale Postion erfasst. Als Verbesserung könnte man zwei um 90 Grad versetzte Drucksensoren verwenden, um die Position im Raum in Erfahrung zu bringen.

Eine Alternative wäre ein Bilderkennungssystem, das die Konturen des Probanden erkennt und so die Positionen der einzelnen Gliedmaßen berechnet. Leider sind Bilderkennungssysteme relativ rechenintensiv und nicht ganz trivial, so dass wir von dieser Idee Abstand nahmen. Der Effekt auf einer Bühne dürfte weniger beeindruckend sein.

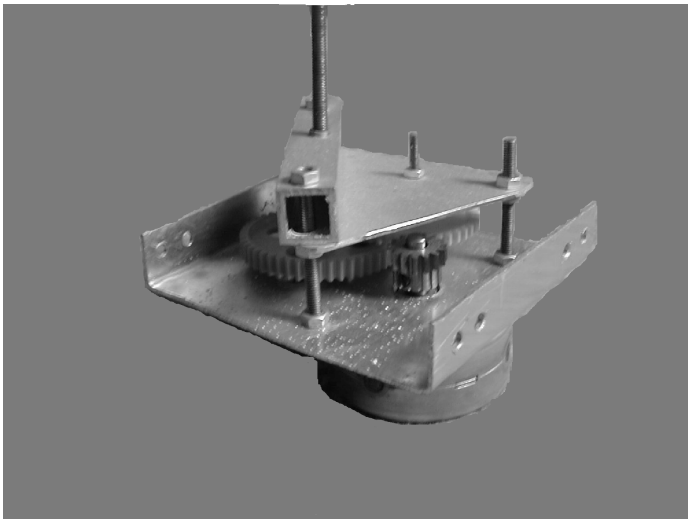

Abbildung 7.8: Marion - Das Getriebe

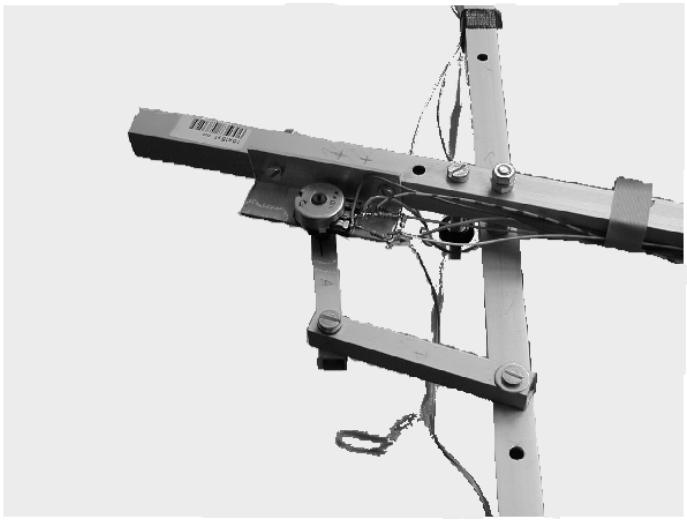

Abbildung 7.9: Eingabegerät: Das Kreuz

7.6 Hwserver

Der Hwserver ist eine Schnittstelle zwischen der Hardware und der Client-Software. Seine Haupteigenschaften sind seine Netzwerkfähigkeit und die Implementierung als Multi-Threaded-Server.

7.6.1 Anforderungen an den Hwserver

Der Server sollte auf ein Netzwerkprotokoll aufsetzen und ein Klartext-Protokoll verwenden, um zu Testzwecken auch mittels Telnet mit diesem zu arbeiten. Es sollte eine Authentifizierung und verschiedene Benutzer und Rechte verwalten können, sowie die Fähigkeit besitzen mehrere Benutzer gleichzeitig bedienen zu können.

Sein Verhalten sollte möglichst dynamisch und seine Benutzung objektorientiert sein. Hardwarekomponenten sollten auf Namen abgebildet werden und über diese ansprechbar sein. Klienten sollten über die Veränderungen des Zustands, der auf dem Server bereitgestellten Objekte in-

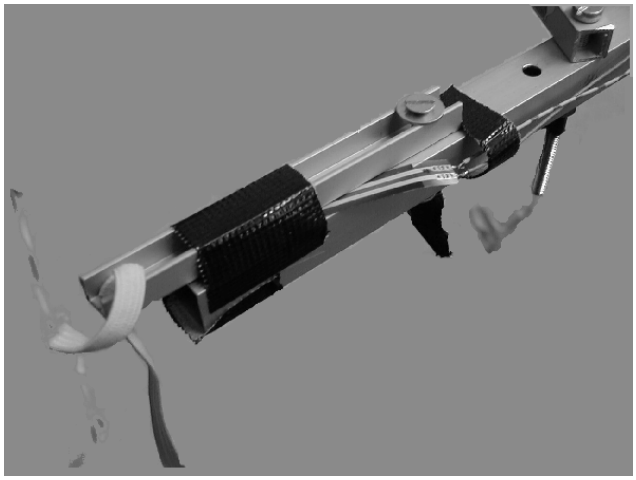

Abbildung 7.10: Schaltplan: Drucksensor

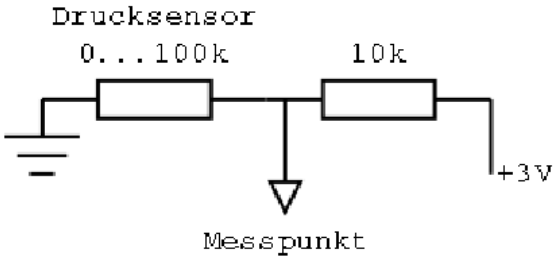

Abbildung 7.11: Spannungsteiler für die Drucksensor

formiert werden, um Polling der Klienten auf dem Server zu vermeiden.

Die Hardwareanforderungen an den Server sollten sich in Grenzen halten. Möglicherweise müsste er auf betagter Hardware zum Einsatz kommen.

Eine Verschlüsselung der Kommunikation war nicht vorgesehen, da der Einsatz in ersten Linie fürs LAN gedacht war und kein produktionsreifes System erstellt werden sollte.

Die Anforderungen an die Zuverlässigkeit des Servers sind hoch. Abstürze und Speicherlöcher während einer Aufführung würden fatale Folgen haben.

7.6.2 Entwicklung, Entwurf und Aufbau des Hwservers

Anfangs gab es keinen genauen Entwurf über den Aufbau des Servers. Um die TCP/IP-Kommunikation wurde der Hardwaretreiber für die Servokarte und die Benutzerauthentifizierung gebaut.

Beim Hinzufügen weiterer Hardwaretreiber zeigte sich, das ein wesentlich klareres Modell mit besserer Trennung von Nöten war.

Die Treiber für die Hardware wurden in Low-Level-Treiber und in Schnittstellenmodule aufgeteilt. Die Low-Level-Treiber sprechen die Hardware direkt an und haben keine Abhängigkeit zum Server. Die Schittstellenmodule für den Hwserver werden wie eine Klasse mit Methoden behandelt, die von einer darüber liegenden Schicht, dem Wrapper, aufgerufen werden. Der Wrapper identifiziert anhand des Objektnamens die zugehörige Klasse des Objekts und die aufzurufende Methode und ruft diese Methode in der Klasse mit den ebenfalls vom Client übergebenen

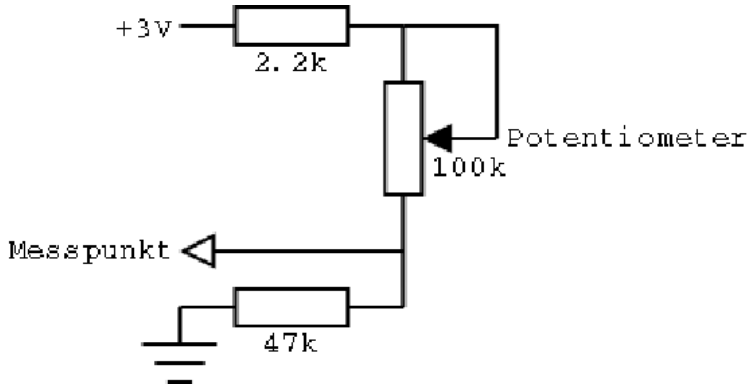

Abbildung 7.12: Spannungsteiler für die Drehungwinkel

Parametern auf.

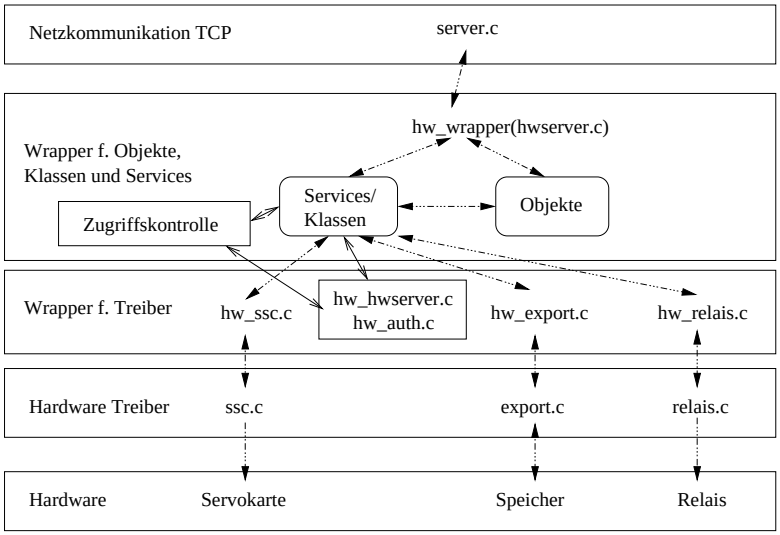

Abbildung 7.13: Schichten des Hwservers

Beim Starten des Hwservers registrieren sich die Klassen-Module(z.B. SSC für die Servohardware) beim Wrapper. Die Module sind zwar statisch zum Hwserver gelinkt, haben jedoch alle die gleichen Schnittstellen. So ist z.B. auch der Hwserver nur ein registriertes Modul und unterscheidet sich nur gering von den Hardware-Modulen. Auf diese Weise ist die TCP/IP-Kommunikations- und Wrapperschicht sehr klein gehalten und umfasst im Wesentlichen nur noch den Wrapper, einen Service der jeden Objektnamen eine Klasse mit eventl. zugehörigen Hardwarekanal(z.B. Kanalnummer eines Servos) zuordnet und einige Schnittstellen zum Authentifizierungsservice.

Damit der Hwserver von sich aus Klienten über Änderungen informieren kann, ohne das die Klienten den Zustand der Objekte auf dem Hwserver kontrollieren müssen, wurde ein Callback-Mechanismus hinzugefügt. Dabei werden Objekte einer Callbackmanagerklasse(CBMGR) erzeugt, welche die Callback-Verbindung zu jeweils einem Klienten regeln. Es können mehrere Objekte mit diesem Callbackmanager verbunden werden, damit der Klient über Zustandsänderungen der Objekte informiert wird.

Damit die Clientseite den Callback-Mechanismus nutzen kann, muss ein CBReceiver, bzw. CB-Server auf dem Client laufen. Dieser benutzt auf der Clientseite ein Serversocket, damit sich der

Hwserver mit diesem verbinden kann und eine aktive Rolle bei der Übermittlung eines Ereignisses einnehmen kann. Der Hwserver ist also in der Lage Kontakt zum Client herzustellen.

Die Netzwerkverbindung wurde möglichst robust gestaltet. So wird zum Beispiel versucht beim Zusammenbruch einer Verbindung, die Verbindung zur Gegenstelle automatisch herzustellen.

7.6.3 Schnittstellen des Hwserver

Der Hwserver hat auf der einen Seite Schnittstellen zur Hardware und auf der anderen Seite die Schnittstelle zu den Clienten. Die Schnittstelle zu den Clienten wird über eine Netzwerkanbindung realisiert.

7.6.3.1 TCP/IP-Netzwerkschnittstelle

Da der Hwserver über eine TCP/IP-Netzwerkschnittstelle verfügt, ist er über dieses Protokoll direkt ansprechbar. Dieses Protokoll ist ein Klartext-ASCII-Protokoll, welches auch Telnetkompatibel ist. Dies hat sich als vorteilhaft für Testzwecke und zum Debuggen erwiesen.

Die Authentifizierung findet durch die Übermittlung eines Benutzernamen/Passwort-Paares statt.

Die auf dem Server ausgeführten Befehle werden vom Server quittiert, bzw. die Ergebnisse werden zum Client übertragen.

7.6.3.2 Java-Anbindung

Für die Implementierung von Java- und VRML-Clients wurden Java-Klassen implementiert. Diese können in Java-Programmen und in VRML-Scripten mit eingebetteten Java-Klassen verwendet werden.

Die Objekte der *Mario*-Klasse stellen die Verbindung zum Hwserver her. Dabei wird eine Liste von IP-Adressen bzw. Namen nach einem laufenden Hwserver mit der gesuchten Marionette durchsucht. Wurde der Hwserver gefunden, so können aus dem instantiierten Objekt der *Mario*-Klasse die gewünschten Körperteile geholt werden. Diese sind Instanzen der Klasse *Actor*.

Die Klasse *CBServer* ist ein TCP/IP-Server der auf einem Serversocket auf der Klient-Seite läuft. Die Adresse und der Port dieses Sockets werden beim Hwserver registriert und als Verbindung für Callbacks, die vom Server initiiert werden, benutzt. Der CBServer ist als eigener Thread ausgeführt, damit der Client nicht blockiert wird.

Die Klasse Actor ist von der Java-Klasse Observable abgeleitet, was es ermöglicht, das diese Objekte ihre Änderungen anderen Objekten mitteilen können, die das Interface Observer implementieren und sich als Observer für die jeweiligen Actor-Objekte angemeldet haben.

Bei einer Änderung eines zugehörigen Objektes auf dem Hwserver, wird dann die *update*-Methode des Observers ausgeführt. Auf diese Weise lassen sich die Zustände auf dem Server mit denen der virtuellen Modelle synchronisieren.

7.6.3.3 C++-Anbindung

Für die Kommunikation mit dem MCP und der, nicht realisierten Bewegungsverdoppelung über Bilderkennung, wurden C++-Schnittstellen zum Zugriff auf den Hwserver entwickelt und implementiert. Dabei wurden Low-Level-C-Routinen in C++-Klassen gekapselt.

Die C++-Klasse *Mario* sucht anhand einer Liste bestimmte Computer nach einem lauffähigen Hwserver und stellt eine Verbindung her. Aus den Objekten, die von dieser Mario-Klasse instantiiert werden, können dann die gewünschten Servos geholt werden, die von der Klasse Actor instantiiert werden.

Für die C++-Klassen wurde das gleiche Design und der gleiche Entwurf verwendet, der auch für die Java-Klassen entwickelt wurde.

7.6.4 Protokoll des Hwservers

Als Beispiel für das ASCII-Klartext-Protokoll sei folgendes aufgeführt:

```
HWSERVER.login admin geheim
SSC.new 0 MarionRumpf
MarionRumpf.SetLimits 0 180
MarionRumpf.SetPosition 90
CBMGR.new marioseq
MarionRumpf.SetObserver marioseq
ADDA.new 2 ADMarionArmLinks
ADMarionArmLinks.SetObserver marioseq
marioseq.connect theater2 4242
HWSERVER.logout
```

Das Objekt *HWSERVER* ist der Server selbst. In diesem Beispiel meldet sich ein Benutzer(admin) mit einem Passwort(geheim) am Server an.

Es wird ein neues Objekt der Klasse *SSC(serial servo controller)* erzeugt. Dieses Objekt bekommt den Namen *MarionRumpf*. Dieses Objekt benutzt den 0ten Kanal der Karte.

Mit der Methode *SetLimits* werden die Minimal- und Maximal-Werte des Servos vorgegeben. Durch die Methode *SetPosition* wird das Servo in die Position 90 gebracht, was in diesem Fall die Mitte des erlaubten Servoweges ist.

Die Klasse *CBMGR(Callback-Manager)* ist eine Klasse von der Verbindungsobjekte instantiiert werden können, die zur Verbindung mit Klienten verwendet werden können. In diesem Beispiel wird ein Objekt zur Kommunikation mit dem Sequenzer angelegt(marioseq).

Das zuvor angelegte Objekt *MarionRumpf* wird mit den CBMGR *marioseq* verbunden, so das alle Änderungen von *MarionRumpf* über das Objekt *marioseq* gesendet werden können.

ADDA ist eine Klasse, die zur Ansteuerung der AD-Wandlerkarte verwendet wird. Hier wird das Objekt *ADMarionArmLinks* erzeugt, welches den 2ten Kanal der Karte zugeordnet bekommt.

Auch Änderungen dieses Objektes werden über den Sender *marioseq* weitergereicht.

Das Objekt *marioseq* ist noch nicht mit einem Klienten verbunden. Die Verbindung wird erst mit der angebotenen Methode *connect* hergestellt. Dabei wird der Rechnername, bzw. seine IP-Adresse und der Port angegeben, auf dem der Callback-Receiver des Clients wartet.

7.6.5 Implementierung

Der Hwserver wurde in C implementiert. Da wir die Verwendung der Software auf schlecht ausgestatteter Hardware nicht ausschliessen wollten, haben wir diesen kleinsten gemeinsamen Nenner gewählt.

Es wurde eine objektorientierte Programmierung in C angewendet.

7.6.6 Diskussion

Die anfängliche Festlegung auf die Ansteuerung der Servos prägte zu Beginn den Entwurf und das Design. In mehreren Zyklen wurden Entwurf, Design und die Implementierung jedoch verfeinert.

Die objektorientierte Programmierung wurde nicht von Anfang an verwendet. Dies erklärt sich durch den wachsenden Umfang und die unvollständige Anforderungsanalyse.

Im Laufe des Projektes wurde der Programmcode mehrfach überarbeitet und modularisiert.

Die Stabilität des Programmes hat ein ausreichendes Maß erreicht. Abstürze kamen, bei der für die Vorführung eingesetzten Version, sehr selten vor. Dies bedeutet jedoch auch, das das Programm nicht fehlerfrei ist.

7.7 Client zum Hwserver

Der Client liest die Eingaben von der Standard-Eingabe(stdin) des Computers und übertragt diese Eingaben an den Server. Die Rückmeldungen des Servers werden auf der Standardausgabe(stdout) des Clients ausgegeben. Startet man den Server ohne die Ein- und Ausgabe umzulenken, so verhält sich der Client ähnlich wie ein Telnet-Client.

Der Client wird hauptsächlich von den Initialisierungs-Skripten gebraucht, die ihre Befehle an den Server über eine Pipe an diesen Client senden.

Dieser Client wurde anfangs, durch Umlenken der Ein- und Ausgabe, auch für die Kommunikation zwischen den Sequenzer und dem Hwserver benutzt. Später wurde er beim Sequenzer durch neue C-Funktionen ersetzt, wodurch eine bessere Kontrolle des Kommunikationsflusses ermöglicht wurde.

Da der Kommandozeilen-Client im wesentlichen zu Testzwecken und in der Anfangsphase gebraucht und gepflegt wurde, zeichnet er sich durch kein gutes Design aus und besitzt auch keine hohe Qualität. Das war nicht tragisch, da auf Applikationsebene später fast ausschliesslich die neueren und besseren APIs Verwendung fanden.

7.7.1 C-Module zur Verwendung in Clients

Unabhängig vom Hwserver-Client, wurden im Verlauf des Projektes wesentlich saubere C-Module implementiert, die für die Kommunikation mit dem Hwserver zuständig sind.

Dazu gehören Module für das clientseitige Steuern des Servers und für den clientseitigen Callbackserver.

7.8 MarioSeq

MarioSeq ist ein Sequenzer, der zum Aufzeichnen und Abspielen von Sequenzen für die Marionette dient. Er verfügt auch über eine Schittstelle mit der er vom MCP gesteuert werden kann.

Die Implementierung eines Sequenzers war von Anfang an vorgesehen und es gab schon eine Version die in Perl geschrieben war. Diese Version lief im Text-Terminal wurde aber nie ganz fertig gestellt.

Nachdem Tim das Projekt verlassen hatte, beschlossen wir mit der Implementierung eines Sequenzers von vorne zu beginnen. Die neue Version sollte jedoch eine Grafik-GUI besitzen.

7.8.1 Anforderungen

Der Sequenzer sollte das Abspielen und Aufzeichnen von Bewegungssequenzen für die Marionette ermöglichen. Es hat sich nämlich im praktischen Betrieb gezeigt, das eine interaktive Eingabemöglichkeit zum Entwickeln einer Bewegungssequenz unverzichtbar war.

Das Laden, Speichern und Editieren von Sequenzen sollte unterstützt werden. Auch mussten die Callbacks des Hwservers behandelt werden und die Anzeige der aktuellen Servostellungen aktualisiert werden.

Das MCP sollte in der Lage sein den Sequenzer mit Sequenzen zu laden und diese abspielen zu lassen.

Zudem sollte es möglich sein auszuwählen, welcher Client die Kontrolle über die Marionette hat.

7.8.2 Entwurf und Entwicklung

Der Sequenzer wurde als Rapid Applikation Development entwickelt. Mit Glade wurde die Oberfläche zusammengebaut und in mehreren Zyklen überarbeitet. Die von Glade generierten C-Funktionsrümpfe wurden mit Emacs um Funktionalität bereichert.

Bei den Proben und Tests wurden die Anforderungen und Änderungswünsche eingebracht und es begann ein weiterer Entwicklungszyklus. Der Sequenzer ist eher ein funktionsfähiger Prototyp, der in kurzer Zeit entstanden ist.

Die Schichten des Programmes sind nicht klar genug getrennt, so das die Oberfläche eng mit der Funktionalität verknüpft ist und schwer davon zu trennen ist. Dies brachte unter anderen die Verwendung des Glade-GUI-Builders mit sich.

Eine klare Trennung zwischen der Sequenzerfunktionalität und der Bedienungs-/MCP-Fernsteuerungsschicht wäre ein eleganteres Design gewesen. Ein solches Design hätte man als Client/Server-Anwendung oder als Model/View/Controller-Anwendung realisieren können.

7.8.3 Implementierung

Der Sequenzer wurde in C implementiert und benutzt das Gtk+ GUI Toolkit. Obwohl das Gtk+ Toolkit Threadsicher ist, musste bei der Implementierung sorgfältig auf Nebenläufigkeit geachtet werden, da es mitunter zu Dead-Locks und anderen nebenläufigen Problemen kam. Es gibt ein globales Lock für die Gtk+-Oberfläche.

Trotz der Verwendung des Oberflächengestaltungtools Glade, ist ein gutes Wissen über das Gtk+-Toolkit unverzichtbar. Die Gtk+-Dokumentierung weist leider noch ein paar Lücken auf. In der Praxis ergaben sich jedoch dadurch keine Probleme. Obgleich Gtk+ nur in C programmiert wird, ist es objektorientiert und verfügt über ein Signal/Ereignis-Mechanismus. Zur Laufzeit des Systems werden die Objekte, die Signale und die zugehörigen Callback-Methoden der Oberfläche initialisiert. Danach verharrt das Hauptprogramm in einer Ereignisschleife. Werden Ereignisse von der Oberfläche ausgelöst, z.B. durch Interaktion mit den Benutzer, so werden die Callback-Funktionen mit einer Referenz auf das ausgelöste Objekt und dem Signal aufgerufen.

Das Gtk+-Toolkit besteht aus drei Komponenten:

glib: Es beinhaltet Datentypen, Macros und Funktionen für diese Datentypen, Stringbehandlung, Datenverwaltung und anderes. Die glib besitzt keine Grafikfunktionen.

gdk: Ein Wrapper zu den Low-Level-Grafikfunktionen des Zielsystems. Es ist ein Bestandteil vom gtk+-Paket.

gtk+: Eine Sammlung von Bedienungselementen für die grafische Benutzerschnittstelle. Darunter auch einige Komponenten.

Glib und Gdk/Gtk+ unterliegen der LGPL-Lizenz. Sie sind einerseits frei und mit Quellcode erhältlich, andererseits können sie mit Programmen gelinkt werden, ohne das die anderen Programme die gleiche Lizenz annehmen. Dies führt dazu, das dieses Toolkit auch gerne von kommerziellen Anbietern benutzt wird.

Da der Sequenzer leider nicht klar von der GUI trennbar implementiert worden ist, kann die Sichtbarkeit der GUI durch das MCP ein- und ausgeschaltet werden, wobei die GUI auch in Verbindung mit dem MCP immer mitläuft.

Glade befindet sich noch in Entwicklung und besitzt somit Beta-Status. Es ist jedoch nur ein Fehler als störend aufgefallen, bei dem Glade bei der Erzeugung der Funktionsrümpfe schon vorhandene Funktionsrümpfe nicht erkannte und als leere Funktionsrümpfe ans Ende der Quellcodedatei angefügt hat. Es hat jedoch keine Daten überschrieben und das Problem war durch Löschen der angehängten leeren Funktionsrümpfe beseitigt. Glade unterstützt noch nicht alle Möglichkeiten der Widgets des Gtk+, weshalb ein benötigter Widgettyp per Hand in das Programm integriert werden musste.

Andere Fehler oder gar Abstürze haben wir bei Glade nicht provozieren können oder blieben unbemerkt.

Es fehlt bei Glade z.Zt. noch die Möglichkeit eigene Widgets aus mehreren Anderen zusammenzusetzen und diese in einem Programm mehrmals zu verwenden. So waren wir gezwungen für alle Servos die Kontrollelemente mehrmals anzulegen.

Ausserdem ist Glade noch keine IDE. Es gibt keine Anbindung an einen Editor. Der verwendete Emacs-Editor mit diversen Erweiterungen, kam allerdings einer IDE schon sehr nahe.

7.8.4 Schnittstellen

Der Sequenzer hat Schnittstellen zum Hwserver, zum MCP und eine grafische Benutzerschnittstelle zur Bedienung des Programms.

7.8.4.1 C++- Schnittstelle zum MCP

Über die Schnittstelle zum MCP kann der Sequenzer ferngesteuert werden.

Die Kommunikation mit dem MCP geschieht mit FIFOs/Named Pipes im lokalen Filesystem. Dies hat zur Folge, das der Sequenzer auf dem gleichen Rechner laufen muss wie das MCP. Dieser Umstand war aber gegeben. Eine Kommunikation über das Netzwerk durch Benutzung von TCP/IP wäre ebenfalls einfach zu realisieren gewesen, da die notwendigen Module für die Realisierung der Callback-Implementierung des Hwservers bereits entwickelt wurden, und sehr flexibel und einfach einzusetzen sind.

Eine sinnvolle Ergänzung, die in Erwägung gezogen wurde, aber aus Zeitgründen nicht zur Realisierung kam, ist das Absenden von Nachrichten an das MCP zwecks Synchronisierung. Da in der Praxis der Theatervorführung der Sequenzer jedoch nicht das Timing vorgab, wurde diese Möglichkeit auch nicht vermisst.

Auf der MCP-Seite wird die Verbindung mit der C++- Klasse MarioSeq hergestellt. Diese Libary gehört zum Hwserver/mariolib-Paket.

7.8.4.2 TCP/IP-Schnittstelle zum Hwserver

Über die in C geschriebenen Client-Bibliotheken wird die TCP/IP Verbindung zum Hwserver hergestellt.

Die Befehle werden über eine TCP-Socket-Verbindung an den Server gesendet. Ein weiterer Thread wartet an einer Serversocket auf Callbacks vom Server.

7.8.4.3 Benutzerinterface

Für den Benutzer, der das Programm interaktiv steuert, ist eine grafische Benutzeroberfläche vorhanden. Von dieser kann z.B. auch gesteuert werden, welches System die Kontrolle über die Marionette hat. Mit ihr werden auch die Servos der Marionette kalibriert und es kann eine Eichung auf den "Galgen" vorgenommen werden.

Anhand von Reglern kann die Stellung der Servomotoren verändert werden. In einer editierbaren Liste werden die Positionen der Servomotoren zu einer Zeitachse angezeigt.

Es ist möglich die Sequenz zu bearbeiten, den zeitlichen Abstand zwischen zwei Schritten zu ändern und einen beliebigen Zustand der Sequenz auf der Marionette darzustellen.

Menüpunkte zum Laden, Speichern und Zurücksetzen der Konfiguration sind ebenfalls vorhanden.

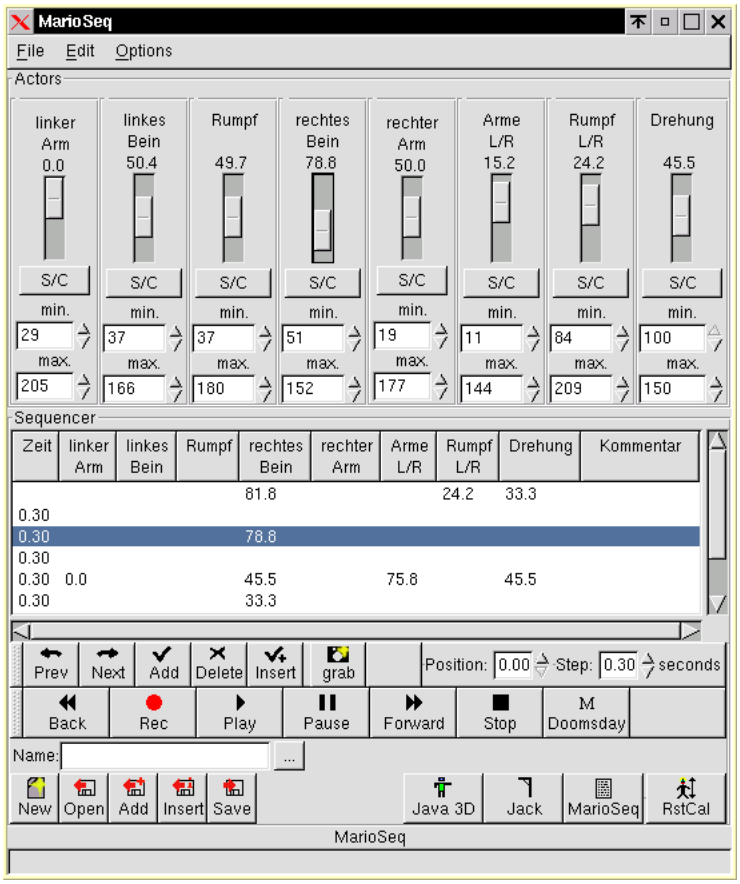

Abbildung 7.14: Der Sequenzer mit Gtk+-Benutzerinterface

7.8.5 Eigene Bewertung und Schlussfolgerung

Im Nachhinein wäre die Verwendung von C++ als Implementierungssprache möglicherweise eine bessere Wahl gewesen. Eine Sprachbindung für Gtk+ ist für C++ vorhanden.

Als Alternative zum Gtk+ Toolkit wäre auch das sehr hochwertige Qt-Toolkit denkbar gewesen. Für nicht-kommerzielle Anwendungen war es zur Laufzeit des Projektes für X-Windows-Systemen frei verfügbar, später wurde es sogar unter GPL-Lizenz gestellt. Allerdings wäre man dann an C++ oder Python als Programmiersprache gebunden.

Sowohl Gtk+- als auch Qt-basierte Programme sind auch auf Windows-Plattformen übersetzbar.

7.8.6 Verwendete Tools und Bibliotheken

Als Grafiktoolkit kam das Gimp Toolkit(Gtk+) in der Version 1.2.x zur Anwendung. Für Gtk+ gibt es eine große Zahl von Sprachbindungen für folgende Sprachen(Stand 15.09.2000): Ada95, C++, C, Eiffel, Haskell, Pascal, Perl, Emacs-Lisp, Guile, Objective C, Java Script, Dylan, Objective Caml, Pike und TOM. Gtk+ wurde ursprünglich als Grafiktoolkit für das Bildbearbeitungsprogramm gimp entwickelt, ist mittlerweile jedoch eines der wichtigsten Toolkits für Grafikoberflächen von X-Window-Systemen in der UNIX/Linux-Welt. Bei diesem Grafiktoolkit handelt es sich um ein reifes Produkt, das auch für kommerzielle Zwecke gut einsetzbar ist. Es gibt Tutorien und sehr viele mit diesem Toolkit verwirklichte Programme, die mit Quellcode im Internet zu bekommen sind.

Zur Oberflächengestaltung wurde Glade in den Versionen 0.5.7/0.5.8/0.5.9 verwendet. Glade ist ein visuelles Toolkit zum Designen von Gtk+ Oberflächen. Dabei werden viele Sprachbindungen unterstützt: C, C++, Perl, ADA 95 und Eiffel. Glade ist keine Entwicklungsumgebung, sondern nur ein visuelles Tool zum Erstellen einer GUI und erzeugt nur die Funktionsrümpfe für die Callbacks, die vom Gtk+ Toolkit initiiert werden.

Anfangs wurde für die Implementierung der TCP/IP-Übertragung in den C++-Bibliotheken die APE-C++-Netzwerkklasse verwendet. Da jedoch Programme, die diese Bibliothek und zugleich Pthreads verwendeten unerwartet abstürzten, wurde auf die Verwendung dieser Bibliothek verzichtet. APE hat eine eigene Thread-Klasse. APE ist heute Bestandteil der Common C++-Bibliothek und nicht mehr im Betastatus.

Vorteile von APE/Common C++ ist die Verfügbarkeit auf verschiedenen Plattformen wie Linux, *BSD, Solaris und Win32. Jedoch glückte die Complilierung von APE unter Windows nicht.

Der GNU-C-Compiler ist ein sehr hochwertiger Compiler. Er unterstützt eine sehr grosse Anzahl von Zielsystemen und kann auch als Cross-Compiler verwendet werden. Der Compiler selbst steht unter GPL.

Um das Compilieren und Installieren des Sequenzers zu vereinfachen, wurde *Autoconf* und *Automake* verwendet. Autoconf erzeugt das Konfigurationsskript *configure*, welches vor dem Compilieren des Pakets ausgeführt wird und automatisch für das Zielsystem passende Makefiles erzeugt. Das Erzeugen der Makefiles auf dem Zielsystem geschieht mit Hilfe von Automake. Das Skript *configure* wird entsprechend der Abhängigkeiten des Pakets erstellt.

Autoconf wird von Glade unterstützt und bereitete keine Probleme.

Als Editor wurde GNU-Emacs verwendet. Zusätzliche Erweiterungen verwandelten den Editor in eine leistungsfähige IDE.

Da keine kommerzielle Anwendung des Marionetten-Teilsystems vorgesehen war, wäre auch das Erben der restriktiveren GPL keine wirkliches Problem. In der endgültigen Lösung ist jedoch dieser Fall nicht aufgetreten, so das auch eine Weitergabe ohne Quellcode möglich wäre.

Bei der Marionette wurde ausschliesslich Software verwendet, die ohne Lizenzgebühren verwendet werden darf. Die Qualität der verwendeten Komponenten erwies sich als gut bis sehr gut.

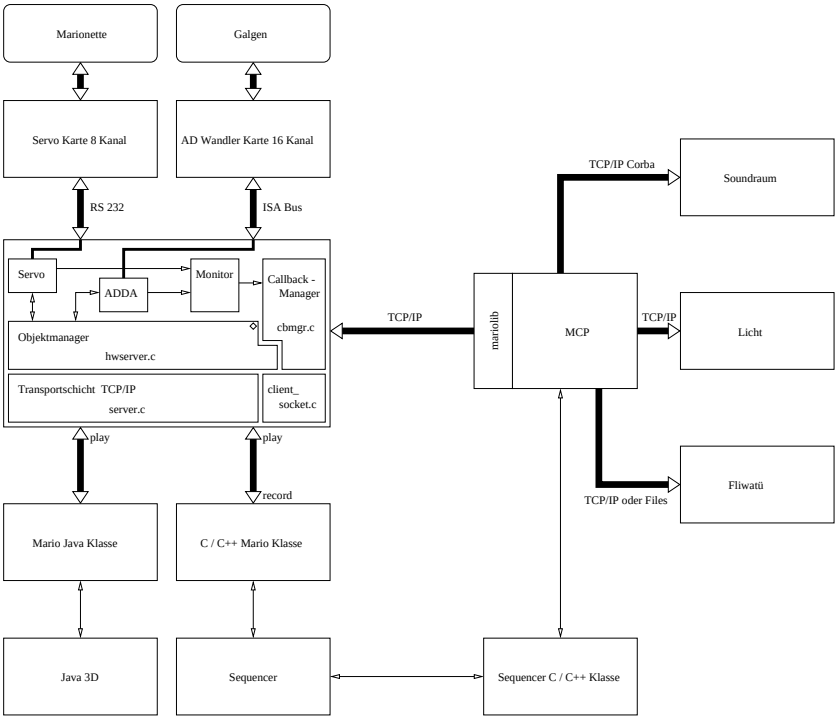

Abbildung 7.15: Gesamtarchitektur

Kapitel 8

Von der Wurst zum Kubismus

8.1 Einleitung

Zur Beginn des Projektes gab es eine zeitliche Periode namens Ideensammlung. In der Zeit wurde versucht verschiedene Ideen zu sammeln und anschließend diese auf die Realisierungsmöglichkeit zu prüfen. Mir schwebten sehr viele interessante Projekte vor, aber leider musste ich kurze Zeit danach feststellen, dass die Realisierung der Projekte mit dem mir vorhandenen Ressourcen nicht möglich wären wie z.B ein intelligenter Kühlschrank, der dem Benutzer über das Web über seinen Inhalt Informationen mitteilt.

Da ich mich für die Darstellung der virtuellen Welt sehr interessiert habe, sah ich jetzt in dem Projekt die Chance kommen, mich mit der Thematik auseinander zu setzen. In einem Gespräch mit den Mitgliedern der Gruppe Würstchenbude gelang es mir, sie zu überzeugen, die Würstchenbude um eine virtuelle Welt zu erweitern.

8.2 Würstchenbude

8.2.1 Grundidee

Die Gruppe Würstchenbude umfasste vier Personen. Nachdem wir eine Performance des Künstlers Stelarc in Hamburg besucht haben, wurde die Gruppe von der Idee der Performance inspiriert, um ein Produkt mit pneumatischen Elementen zu realisieren.

8.2.2 Anforderungsdefinition

In einer Würstchendose befindet sich ein Roboter, dessen Kopf eine Kamera darstellen soll, welche mittels Sensorien aktiviert wird, um die Zuschauer zu fotografieren. Die Bilder sollen dann auf einer Leinwand karikaturistisch projiziert und gleichzeitig die Szene in einer virtuellen Welt simuliert werden.

Mit einem Datenhandschuh können die Zuschauer den Roboter steuern. Der Zuschauer soll an den Stand des Roboters mittels einer Aktion angelockt werden, so dass die Position des Zuschauers optimal im Fokus der Kamera ist.

Der Roboter soll aus vier Kernbestandteilen bestehen: An einem geometrischen Körper sollen zwei pneumatische Zylinder befestigt werden, deren Funktion darin besteht, den Roboter aus der Würstchendose zum Erscheinen zu bringen .

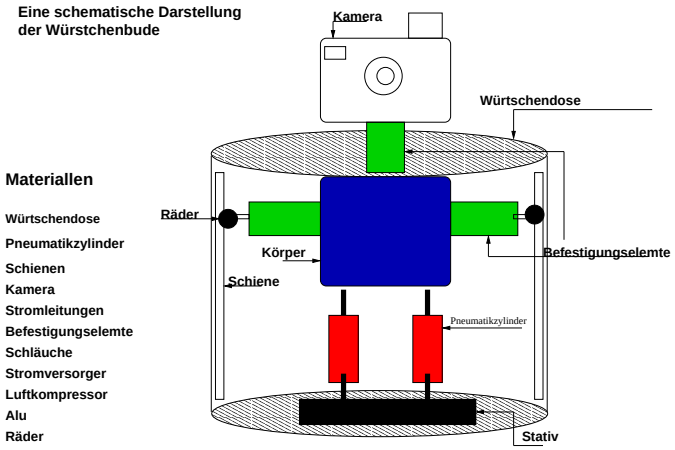

Abbildung 8.1: Würstchenbude

8.2.3 Vorgehensweise

Wir haben uns am Anfang über die Realisierungsmöglichkeiten und deren Umsetzung Gedanken gemacht und versucht, das nötige Fachwissen uns anzueignen.

Das grobe Konzept, welches im Plenum vorgestellt worden war, wurde verfeinert und um eine virtuelle Welt erweitert.

Somit hatten wir jetzt verschiedene Gebiete, die jedem nach Interesse in seine Obhut gelegt wurde. Dies schloss aber eine übergreifende Arbeit miteinander nicht aus.

So konnten z.B. Mitglieder der Gruppe sich jederzeit anderen Themen innerhalb der Gruppe anschließen.

8.2.4 Komponenten der Würstchenbude

8.2.4.1 Pneumatik

Das Grundwissen konnten wir uns in der Veranstaltung von Hermann aneignen. Wir bräuchten zwei Pneumatikzylinder, welche die Aufgabe hatten den Körper des Roboters aus der Dose hinauf und hinab zu bewegen. Zusätzlich sollten weitere Komponenten dazukommen wie z.B. Pneumatikventile, Schläuche, Luftkompressor usw.

8.2.4.2 Elektronik

Hier sollte das Gesamtkonzept aus der Sicht der Elektronik näher untersucht werden.

Welche elektronische Bauelemente brauchen wir?

Wie soll die Integration der Bauelemente stattfinden?

Wie sind die Schnittstellen zu den anderen Derivaten (Datenhandschuh, Kamera, Virtuelle Welt)?

8.2.4.3 Datenhandschuh

Das Hard- und Software wurde uns von Artec zur Verfügung gestellt. Ferner bedienten wir uns hinsichtlich der Schnittstellen und weiteren Problematiken einer uns zur Verfügung gestellten Diplomarbeit.

8.2.4.4 Virtuelle Welt

Hier haben wir uns für die Szenenbeschreibungssprache VRML (Virtual Reality Modelling Language) entschieden.

8.2.4.5 Kamera

Eine Digitalkamera sollte mittels einer Schnittstelle zum PC die aufgenommenen Bilder an den Server senden, welche anschließend auf einer Leinwand karikaturistisch projiziert werden sollten.

8.2.4.6 Schnittstellen

Zu definierenden Schnittstellen:

Es sollten insgesamt vier Schnittstellen definiert werden. Die Kommunikation zwischen den Actoren (Clients) und dem Server sollte über das TCP/IP Netzwerkprotokoll durchgeführt werden.

Server - Würstchendose

Server - virtuelle Welt

Server - Datenhandschuh

Server - Kamera

8.2.5 Resultat

Um die Realisierungsmöglichkeiten besser einschätzen zu können, wurden die ersten Versuche mit Fischertechnik durchgeführt. So wurden wir auf einige physikalische Komplikationen aufmerksam. Diese Probleme beinhalteten unter anderem die senkrechte und waagerechte Bewegung des Roboters.

Während der Problemlösungsphase schrumpfte die Gruppe auf zwei Personen, da dieses Vorhaben nicht mit zwei Personen zu bewältigen wäre, entschieden wir uns für eine Auflösung der Gruppe.

8.3 VRML-Marionette

Da ich mein Wissen im Bereich Darstellung virtueller Welten weiter vertiefen wollte, entschied ich mich für eine Zusammenarbeit mit der Marionetten-Gruppe.

Die reale Marionette sollte in einer virtuellen Welt dargestellt werden, welche in Echtzeit gesteuert werden sollte.

8.3.1 Anforderungsdefinition

- Darstellung der realen Marionette in einer virtuellen Welt

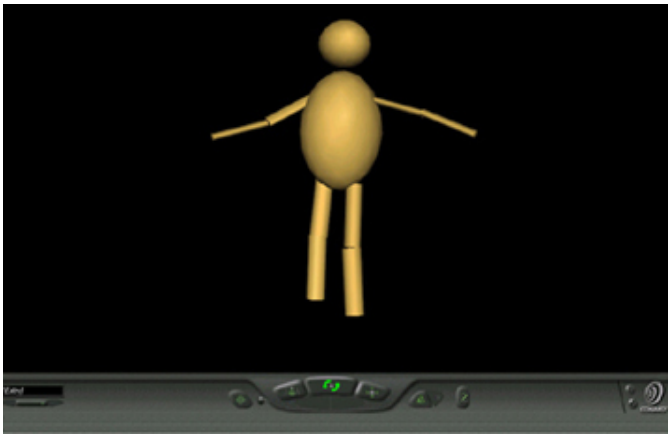

Abbildung 8.2: Simulation des VRML-Männleins im Cosmoplayer unter Netscape

- Einbindung der virtuellen Welt in HTML, um ein Mehrbenutzerwelt im World Wide Web zu ermöglichen
- Interaktion mit dem User
- zeitgleiche Darstellung der Änderungen:
  - Änderung durch programm- oder benutzergesteuerte Objekte:
  - ändert ein Objekt seine Position, muss dies in allen Sichten erkennbar sein.

8.3.2 Machbarkeitsanalyse

Zunächst einmal musste ich alle 3D-Darstellungstechniken, die unsere Anforderungen hinreichend erfüllen sollten, untersuchen und experimentieren.

Weiterhin war die Frage zu klären, welche Tools zum Einsatz kommen könnten, um die Realisierung effizient zu erleichtern.

8.3.3 VRML

VRML(Virtual Reality Modelling Language) ist eine Szenenbeschreibungssprache. Konzipiert wurde sie als systemunabhängige Ergänzung von HTML. Sie basiert auf dem Inventor-Format von Silicon Graphics und wurde um zwei Konstrukte, WWW-Inline und WWW-Anker, erweitert.

Mit Hilfe von VRML (außer Java 3D, siehe dazu Kapitel Java 3D) implementierten Welten ist es möglich, virtuellen Welten über das World Wide Web miteinander zu verbinden. Dies ermöglicht eine ortsunabhängige Interaktion zwischen dem User und der virtuellen Welt.

Diese Tatsache führte zur der Entscheidung, vorerst die virtuelle Welt mit Hilfe von VRML darzustellen.

8.3.4 Technische Ausstattung

Während der Arbeit mit VRML stand mir folgende Hard- und Software zur Verfügung. Sie genügen natürlich für Standardsimulationen, bei anspruchsvolleren Anwendungen bedarf es leistungsfähigerer Hard- und Software.

8.3.5 Hardware

AMD K6-200MHz

32Mbyte-RAM

10Gbyte-Festplatte

17-Zoll-Monitor

8.3.6 Software

VRML Plugin(Cosmoplayer für den Browser)

Graphics Engine: OpenGL

8.3.7 Betriebssystem

Windows

Linux

8.3.8 Entwicklungs-Tools

Cosmoworlds Cosmoworld ist ein 3D-Entwicklungstool, mit dem man speziell VRML Welten modellieren kann. In dem Workshop "Kontrollmechanismen von VRML97" (Veranstallter: Artec), modellierte ich mit dem Comsmoworld die Geometrie meiner ersten virtuellen Puppe in VRML.

8.3.9 Protokolle

Die Kommunikation zwischen dem Client(virtuelle Welt) und dem Server(HwServer) erfolgt über das TCP/IP Protokoll.

8.3.10 Resultat

Nach einigen Experimente wurde schnell klar, dass das VRML97 unsere Anforderungen nicht hinreichend erfüllen konnte.

In VRML werden die Dateien im ASCII-Format abgelegt, da aber die Dateien sehr groß sein können, dauert das Laden über das Netz im Browser sehr lange. VRML97 verfügt weder über 3D-Kompression noch über inkrementelles Laden (während des Ladens wird erst ein grobes Modell angezeigt, das dann weiter verfeinert wird).

Anwenderseitig basieren gängige Systeme auf einem Plugin (Cosmoplayer), das in einem Webbrowser läuft und dessen Javaimplementierung mitbenutzt. An dieser Schnittstelle gibt es entscheidende Unterschiede, je nachdem was für einen Webbrowser man verwendet und welches VRML-Plugin man gerade benutzt und unter welchem Betriebssystem.

Das entscheidende Problem war jedoch, dass es für Linux keine vollständige VRML97-Implementierung gab. Windows 98 zeigte bei der Netzwerkimplementierung unter Java unberechenbare und nicht zu erklärenden Probleme.

Da es aber eine Alternative gab, nämlich Java 3D, die unsere Anforderungen optimal erfüllte und ferner selbst eine API von einer Programmiersprache war, entschied ich mich für dieses

'Java 3D'.

Es gibt zwar in Java 3D die Möglichkeit VRML-Dateien mittels Loader zu importieren, jedoch würde dies die eigentliche Sache nicht vorantreiben, da ich auf die grafische Darstellung wenig Wert gelegt habe.

8.4 Java3D-Marionette

Das Java 3D ist eine Standarderweiterung API und der größte Teil der Media und Communication API von Java. Die Java 3D-API baut auf existierende Graphikstandards wie Direct3D, OpenGL, QuickDraw3D und XGL auf und entwickelt sie weiter z.B. durch Einführung von Konzepten und Klassen für die Modellierung von Raumklang (3D spatial sound).

Die wesentliche Ziele von Java 3D sind:

1. abschließende umfangreiche Sammlung von 3D-Klassen, die eine breite Vielfalt von räumlichen Modellen erlauben
2. Unterstützung des objektorientierten Paradigmas auch für die 3D-Programmierung
3. Unterstützung deklarativer und proprietärer Markupsprachen für 3D (z.B. VRML, CAD-Sprachen)

8.4.1 Technische Ausstattung

8.4.1.1 Software

1. Java 2-Platform(mindestens JDK 1.2)
2. die Klassen des API, sowie einige Utility(Installation des Java-3D-API)
3. Grafikbeschleuniger: OpenGL 1.1

8.4.1.2 Hardware

In dem letzten Semester des Projekts stand mir ein neuer Rechner zur Verfügung, der wesentlich leistungsfähiger war als der alte Rechner.

1. Pentium III-600MHz
2. 3D-Voodoo-Grafikkarte
3. 20Gbyte-Festplatte
4. 256Mbyte-Arbeitsspeicher

8.4.1.3 Betriebssysteme

Unter folgenden Betriebssysteme wurde das Programm entwickelt:

1. Linux
2. Windows 98

8.4.1.4 Entwicklungsumgebungen

Als Entwicklungsumgebungen kamen

1. Emacs unter Linux
2. Kawa 3.21 unter Windows 98
3. Suns Java SDK

zum Einsatz.

8.4.2 Entwicklung, Entwurf und Aufbau des 3D-Männleins

Das Java 3D Programm besteht aus der Klasse Callback-Miniserver, MarioUpdate Methode, welche in der Mario Klasse eingebettet sind, der Klasse Actor(abgeleitet von der Java-Klasse Observable) und der Darstellungsmethode.

Der Callback-Miniserver ist ein Socketserver, der die Kommunikation mittels TCP/IP mit dem Hwserver aufrechterhält. Der Callback ist ein Ereignisverarbeitungsprogramm. Es beruht auf zwei Elementen: Dem auslösenden Ereignis (Sender) und der Callbackfunktion (Empfänger).

Tritt ein Event ein, benachrichtigt der Sender die einzelnen Empfänger, und die assozierten Callbackfunktionen werden ausgeführt. Als Identifizierung für die Verbindungen mit dem Hwserver dienen die Adresse und der Port, welche beim HwServer initiiert und registriert werden.

Wird ein Signal (Änderung eines Actor Objektes) in Form eines Strings vom HwServer empfangen, wird dies an MarioUpdate-Methode des Observers weitergeleitet und auf den Actor-Objekten ausgeführt.

Die Darstellungsmethode simuliert die Änderungen in Java 3D, somit werden die Änderungszustände der virtuellen Welt mit der realen Marionette synchronisiert.

8.4.2.1 Geometrie

Alle Actoren sind, geometrisch gesehen, Würfel oder Quader. Das heißt es sind lediglich zwei Klassen, die das gesamte Objekt geometrisch beschreiben, ColorCube und ColorQuad.

In Java 3D werden die Oberflächen über Dreiecke definiert. Die Informationen über die Geometrie der Körper sind Ableitungen von der Klasse Geometry.

ColorCube und ColorQuad verwenden intern ein Objekt der Klasse QuadArray, mit dem die Klasse eine Liste von Rechtecken verwaltet, die mit einem Array von 6 mal 4 Punkten die sechs Seiten des Würfels und des Quaders beschreiben.

8.4.2.2 Schematischer Aufbau des 3D-Männlein im Java Applets/Appletviewer

Um das 3D- Männlein im World Wide Web darstellen zu können, wurde das Programm im Java Applet eingebettet. Dies wurde wie folgt durchgeführt:

- Ableiten einer neuen Unterklasse von java.applets.Applet
- Erzeugen eines Canavas3D-Objekts und hinzufügen in das Panel des Applets
- Erzeugen eines BranchGroup-Objekts als Wurzel des Szenengraphen

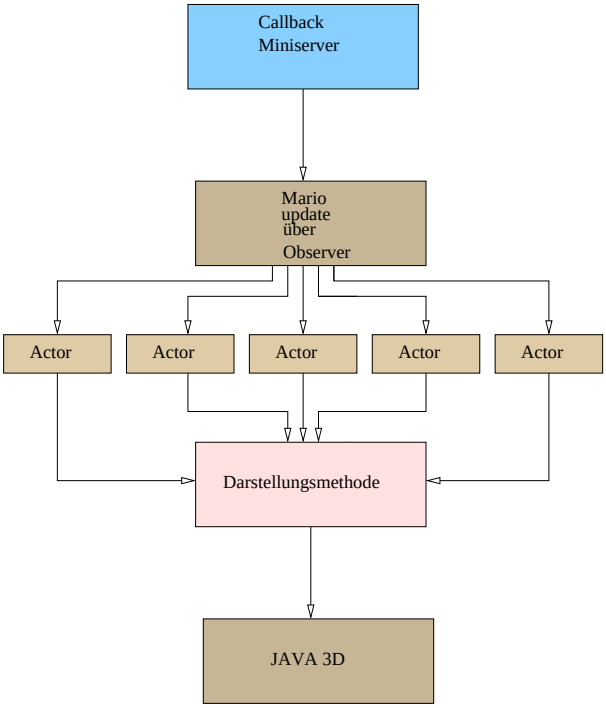

Abbildung 8.3: Java3d-Männlein-Architektur

- Erzeugen einer 3D-Form (Shape3D) mit einem TransformGroup-Knoten als Eltern-Knoten
- Zuordnung des Verhaltens (Behavior) RotationInterpolator zum TransformGroup-Knoten
- Erzeugen des Universums
  - Aufbau des Universums mit einem (HiRes-)Locale-Objekt
  - Erzeugen der graphischen Superstrukturobjekte PhysicalBody, PhysicalEnviroment, View und ViewPlattform
  - Erzeugen eines BranchGroup-Knotens als Wurzel des branch graph der view platform
  - Einfügen des view branch graph in das Locale-Objekt
- Einfügen des Szenengraphen in das Locale-Objekt

8.4.2.3 Wichtige Klassen und Methoden in Mario.java

```
public Mario(String mario, String uname, String passwd)
```

Ein intelligenter Konstruktor, der die Methode searchServer benutzt und versucht damit den Server zu finden. Die Kriterien dabei sind der Name der gesuchten Marionette, der Benutzername für den Server und das Passwort für den Server

```
private void allocActors()
```

Erzeugt Aktoren für die Objekte auf den Server.

```
private final byte[] toBytes(String s)
```

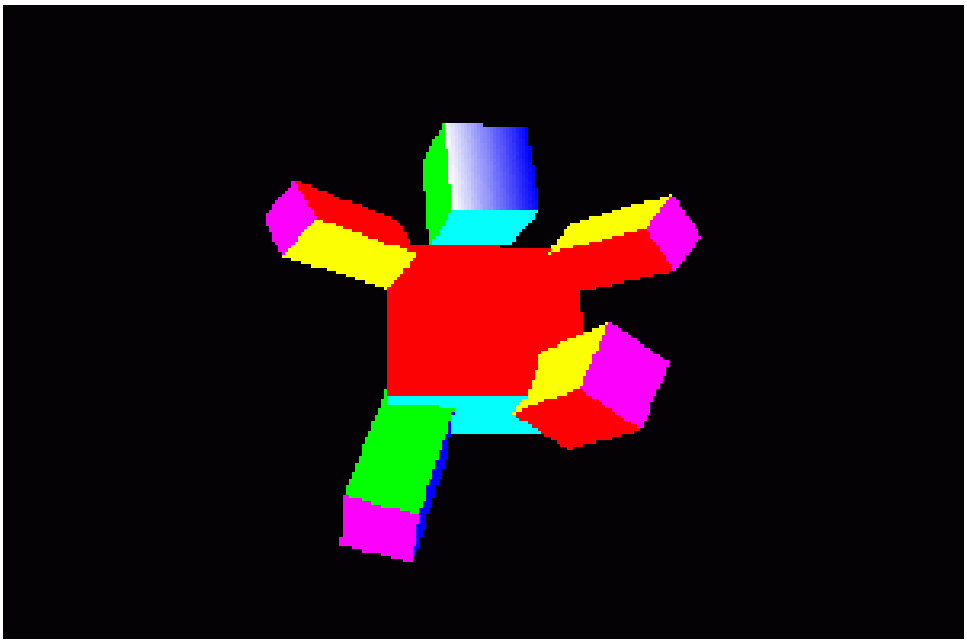

Abbildung 8.4: 3D-Männlein in einem Java Appletviewer

Wandel von String in Byte.

```
public void connect (String server, int port) throws IOException {
s = new Socket(server, port);
```

Stellt eine Verbindung zum Marionettenserver her.

```
public int searchServer(String mario)
```

Versucht den passenden Server zu finden, auf dem das Objekt mario zu finden ist.

```
public Actor getActor(String name)
```

Gibt einen Aktor, der vom Server verwaltet wird.

8.4.2.4 Implementierung

Das 3D-Männlein wurde in Java wechselweise unter Linux und Wndows mit der Entwicklungsumgebung Kawa und Emacs programmiert.

8.4.2.5 Schnittstellen

Der Callback-Miniserver auf der Client Seite bildet die Schnittstelle zum Hwserver. Sie kommunizieren per TCP/IP miteinander.

8.4.3 Ein- und Ausgabegeräte

Ein Entwicklungsziel von Java 3D bestand darin, für die 3D- Anwendungen die Java Philosophie "write once, run anywhere" zu verwirklichen.

Eine Java-3D-Anwendung passt sich ohne Programmänderungen dynamisch an die verschiedenen Geräte an, daher ist sie offen für neue und eigenentwickelte Hardwarekonstruktionen.

8.4.4 Eingabegeräte

Die Maus: Mittels der Maus kann man am Bildschirm die Gelenke der virtuellen Marionette bewegen, dies führt zur der Bewegung der Gelenke der realen Marionette.

Der Sequenzer: Der Sequenzer ist ein Programm, der beliebige Bewegungssequenzen der realen Marionette abspielen und aufzeichnen kann. Siehe auch Kapitel 7 Mario Abschnitt 7.8 MarioSeq.

Der Galgen: Der Galgen ist ein Bändergerüst mit Drucksensoren, welche die Bewegungssignale des Actors erfasst und diese mit der realen Marionette über Hwserver synchronisiert. Die Synchronisation der realen Marionette bewirkt eine Veränderung und damit auch eine entsprechende Synchronisation der virtuellen Marionette. Siehe auch Kapitel 7 Mario Abschnitt 7.5 "Die Galgen-Positionierung im Raum" .

8.4.5 Ausgabegeräte

Der Monitor: Am Monitor wird die Synchronisation der virtuellen Welt visualisiert.

Die reale Marionette(die Servos): Die reale Marionette ist eine gelenkige Holzpuppe, die über sieben Servomotoren vom Computer gesteuert wird. Die reale Marionette kann von der 3D-Welt aus gesteuert werden und umgekehrt kann die reale Marionette die virtuelle Marionette über den MarioSeq steuern. Siehe auch Kapitel 7 Mario Abschnitt 7.4 Marion - Die mechanische Marionette.

8.4.6 Das Starten des Programms

Nachdem der HwServer gestartet worden ist, wird das Java 3D-Männlein Programm gestartet (test2.java). Der Name des Programms test2.java ist zwar nach der Testphase nicht mehr angebracht, allerdings wusste ich bis zum Ende nicht, ob es sich um ein Männlein oder Weiblein handelt. Ich tendierte eher zum Männlein, um aber das andere Geschlecht nicht zu benachteiligen, beließ ich wenigstens den Namen des Programms bei test2.java.

```
java test2.java
```

8.4.7 Resultat

Wenn man das Ergebnis mit der Anforderungsdefintion vergleicht, so wurden alle geforderten Punkte erfüllt.

Die zeitgleiche Interaktion mit der reale Marionette funktioniert einwandfrei. Ferner kann die virtuelle Puppe von den Eingabegeräten Sequenzer und Galgen über die reale Marionette gesteuert werden. Um eine Mehrbenutzerwelt im Web zu ermöglichen, wurde die virtuelle Welt in einem Applet eingebettet.

Obwohl ich mich während des Projekts mit der inversen Kinematik beschäftigt habe und vorhatte, die Funktionalität der virtuellen Puppe um kinematische Bewegungen zu erweitern, nahm ich diesen Punkt nicht in die Anforderungsdefintion auf. Dies hatte mit der Priorität der Entwicklungszyklen zu tun. Da nicht alle Grundfunktionalitäten funktionierten, wie z.B zeitgleiche

Abbildung 8.5: Aufführung des Java 3D-Männleins in einer Interaktion mit der realen Marionette während der Projektpräsentation

Interaktion mit der realen Marionette, verzichtete ich vorerst, ernshaft in Erwägung zu ziehen, die inverse Kinematik mit in die Anforderungsdefinition aufzunehmen (siehe dazu Abschnitt 8.5).

8.4.8 Kinematik

8.4.8.1 Aero: A Physically Based Simulation and Animation System

Die reale Marionette besteht aus mehreren Gelenkkörpern. Um die physikalische Animation der Gelenkkörper in der virtuellen Welt wiedergeben zu können, bedarf es die Einbeziehung Inversen Kinematik (Aero).

Ein Gelenkkörper ist ein System aus gelenkig verbundenen starren Segmenten. Eine effiziente Kontrolle über die Bewegungen erhält man aber erst durch die Einbeziehung der Direkten und Inversen Kinematik und des Begriffs des End-Effektors.

Ein End-Effektor stellt das Endglied einer kinematischen Kette dar und hat oft eine bestimmte Funktionalität.

Während die direkte Kinematik nun die Berechnung der Position und Orientierung eines End-Effektors aus der Vorgabe der Gelenkwinkel beinhaltet, ist es das Ziel der Inversen Kinematik, die Gelenkwinkel zu finden, die zu einer bestimmten Position und Orientierung des End-Effektors gehören. Diese Technik ermöglicht es somit dem Animateur, ein Gelenkkörpermodell über die Spezifikation der End-Effektoren zu steuern.

Aus diesen Gegebenheiten, um die Arbeit zu erleichtern, bietet Aero-Tool die Berechnung bzw. Visualisierung und Simulation von inverser Kinematik.

Es gab zwei Versionen, das Binary-Paket und das mit dem unkompilierten Sourcecode. Am Anfang ließ sich leider der Sourcecode nicht kompilieren, da es mit der Linux Distribution kleine Probleme gab.

Die kompilierte Aero-Version ließ sich jedoch auf SuSE 6.2 installieren. Das FWF Toolkit, welches von Aero benötigt wird (Verion 1.3 bis 1.7), ließ sich auf keinen der Linuxversionen kompilieren, da das FWF-Toolkit nicht mehr gepflegt wurde. Dies führte dazu, dass Aero (unkompilierte Version) sich auf neueren Systemen nicht mehr kompilieren ließ.

Ein Programmteil (src/io.c) von Aero ist in der Lage den aktuellen Zustand des Systems zu speichern. Hier war auch die Schnittstelle für mein Programm.

Solange aber man das eigentliche Programm nicht ändern konnte, war es sehr fraglich sich wegen der knappen Zeit weiterhin mit Aero zu beschäftigen. Die Entscheidung war, dass ich auf Aero bezüglich des Java3D-Männleins leider verzichten mußte.

8.5 Vorlesungen, Seminare und Workshops

Während des zweijährigen Projektes besuchte ich folgende Veranstaltungen:

Vorlesung: Computergraphik (Prof. Frieder Nake)

Um die grundlegende Algorithmen der Computergrafik zu verstehen, ist es zwingend diese Vorlesung zu besuchen.

Workshop: VRML (Artec)

Der einwöchige Workshop vermittelte mir den ersten richtigen Umgang mit VRML Schwerpunkt: Kontrollmechanismen von VRML97

Seminar und Workshop: Pneumatik (Hermann)

In diesem Seminar verbunden mit einem Workshop wurden uns die Grundlagen der Pneumatik in theoretischer und praktischer Form näher erklärt.

Seminar: Sinnenspiel der Medien (Bernd Robben, Prof. W. Bruns)

Zur Anfang verschaffte uns Bernd Robben einen Überblick über das gesamte Spektrum der Sinnenspiele der Medien.

Abschließend wurden unter anderem Aspekte von der Wahrnehmung in Virtual Reality bis soziale Dimensionen von Virtual Reality in Form von Referaten vorgetragen. Ich bearbeitete das Thema *Farbwahrnehmung und Farbdarstellung mit Hilfe des Computers.*

8.6 Schlusswort

Wenn man das Ergebnis betrachtet, dann stellt man fest, dass die zwei Jahre für die Realisierung des 3D-Männleins sehr viel Zeit gewesen ist.

Jedoch habe ich mich in dem zweijährigen Projekt mit sehr interessanten Themen und Techniken befasst, die weit hinaus über die Realisierung des 3D-Männleins gehen.

Angefangen bei Pneumatik, VRML bis Computergrafik, inverse Kinematik und Java 3D. In so einer Zusammensetzung hätte ich nirgendwo diese interessante Themen kennengelernt.

Ich gebe zu, das ich des öfteren überfordert war, aber diese Überforderung stärkte letztendlich den Willen stets am Ball zu bleiben. Meine Vorgehensweise war geprägt von einem Satz: Man muss das Unmögliche versuchen, um das Mögliche zu erreichen!

Kapitel 9

Die Präsentation

In diesem Kapitel wird der *künstlerische* Teil des Projektes beschrieben, das heißt, die Umsetzung der Technik auf die Bühne inklusive der Aufführungen. Es geht also einerseits um den Prozess, der stattfand, als das Projekt noch etwas indifferent war und die Umsetzung noch nicht genau feststand (ähnlich wie die sich ständig ändernde Projektidee), und andererseits um eine kurze Beschreibung der Aufführungen beim Projekttag der Informatik, also der formalen Abschlusspräsentation, und der Aufführung, die wir im Rahmen eines Theaterworkshops in Stuttgart hatten.

9.1 Das inhaltliche Konzept

Inhaltlich geht es in dem Projekt ja um die Auseinandersetzung mit dem Thema Mensch-Maschine-Interaktion. Dazu hatten wir gemäß der ersten Projektidee einige technische Objekte entwickelt. Diese erste Projektidee zu verwirklichen war aber nun mit diesen wenigen Objekten in der restlichen Zeit nicht möglich. Es bestand aber weiterhin die Idee, eine Aufführung zu inszenieren, bei der aber, was erschwerend hinzukam, niemand von uns selber (auf einer Bühne) agieren wollte. Die Maschinen sollten gezeigt werden, die Menschen sollten im Hintergrund bleiben. Inzwischen, also seit dem zweiten Semester bestand eine Zusammenarbeit mit Jörg Richard, einem Professor aus dem Fachbereich Kulturwissenschaft, der uns nicht nur durch die Vermittlung der Historie von Maschinen im Theater Inspirationen lieferte, sondern insgesamt von dem Projekt sehr angetan war und sich stark engagierte.
Die Zusammenarbeit mit ihm und Martin Philipp, einem Kulturwissenschaftsstudenten, den wir über Jörg kennenlerneten, sowie anderen Inspirationen durch ähnlichen Projekten/Künstlern erwies sich als sehr fruchtbar.
Zunächst kam der Gedanke, die Präsentation am Projekttag als Ganzes als Projektpräsentation zu nutzen, das bedeutete, dass nun doch eine theatralische Aufführung zu planen war, wir damit aber gleichzeitig die Sorge loswaren, was wir auf dem Projekttag präsentieren sollten. Die Idee zu einem Theaterstück war geboren.

9.2 Die Bühne

Um ein Theaterstück am Projekttag aufzuführen, brauchten wir also eine Bühne im großen Senatsaal des MZH, denn diese Vorgabe war fest. Wir mussten also eine eigene Bühne entwerfen. Im Gegensatz zur offenen Bauweise des Senatssaales sollte die Bühne eine Art *Blackbox* werden, das heißt ein geschlossener Raum, bei dem jedeR sofort erkennt, was Bühne ist und was nicht, alles auch etwas nach dem Spruch von Brenda Laurel *the stage is all there is.* Größe und Aus-

gestaltung hingen vom noch zu entwerfenden Stück ab.
Da die Bühne keine dauerhafte Installation im MZH sein konnte, musste also erstmal eine Konstruktion entworfen werden, die (in möglich kurzer Zeit) zu einer Bühne werden kann, aber gleichzeitig sehr mobil ist. Dazu benutzten wir Bahnen aus schwarzem Stoff, die wir um Holzlatten wickelten, diese konnten wir bequem im Projektraum lagern und relativ schnell in der Deckenkonstruktion des Senatssaales anbringen, um die gewünschte Blackbox zu verwirklichen, durch Zwischenwände, die wir auch zum Teil schwarz abgehängt haben, war nun aus einem großen zweigeteiltem Raum eine Bühne und ein Zuschauerraum geworden. Für die eigene Bühnenbeleuchtung samt Verkabelung war es schon schwieriger, aber mit etwas Übung durch die Proben war der Auf- und Abbau der gesamten Konstruktion, wie auch der Rechnertechnik zur Steuerung der Objekte bald relativ schnell.

9.3 Proben und Improvisation

Neben diesen technischen Fertigkeiten des Bühnenaufbaus begannen auch bald echt Proben auf dieser Bühne. Die erste Idee, von Martin, bestand darin zunächst einmal zu improvisieren mit Mensch und den Maschinen. Es gab nun also doch eine handelnde Person (hier soll bewusst der Begriff Schauspieler vermieden werden), die von Oliver bzw. Frank dargestellt wurde, als Erst- und Zweitbesetzung. Anhand dieser ersten Versuche entstand das Bild eines Bastlers, der mit seinen Maschinen zusammenlebt, meistens ganz normal, manchmal von ihm bestimmt, manchmal durch das Eigenleben der Geräte.
Die komplette Handlung sollte so mit der Zeit entstehen, ein Probenterminplan wurde ausgearbeitet. Außerdem bestand schon zu diesem Zeitpunkt Kontakt zu einem Cellisten der Bremer Kammerphilharmonie, es war aber noch nicht klar, wie er in das Stück eingebunden werden sollte.

9.4 Das Skript

Während diese ersten Proben vor allem unter der Leitung von Martin stattfanden, kam bei einem Projektplenum nun wieder Jörg Richard dazu und präsentierte ein von ihm entwickeltes Drehbuch, zunächst als Entwurf gedacht, aber schon mit einer starken Festlegung. Einerseits auf die Handlung, die wir ja erst entwickelten beim Improvisieren, andererseits auf die Handelnden, der Cellist rückte in den Vordergrund als handelnde Person, hinzu kam DIE FIGUR, die einer von uns spielen sollte (auf Vorschlag von Jörg und im Hinblick auf den Termin in Stuttgart, der aber noch nicht so fest war, Frank). Ebenso ändert sich nun auch etwas die Thematik, es geht um den Gesamtzusammenhang Mensch-Maschine-Natur, vergleichbar mit den EXPO-Motto.
Diese neue Richtung wurde aber ohne Kritik von allen aufgenommen, nicht zuletzt wegen Zeitdrucks und weil wir nun ein fertiges Konzept hatten, auch wenn es etwas *von außen* kommt und nicht mehr sehr viel mit der Ursprungsidee zu tun hatte. Dies war aber nun etwas Handfestes. Wie die konkrete Umsetzung aussehen würde, hatte noch niemand so recht gesehen, auch die ausgesuchten Zitate, die an verschiedenen Stellen im Stück vorkommen (Botho Strauß), wurden ohne weitere Diskussion übernommen.
Nun begann der harte Probenteil, der aufgrund der Termine von Marc, dem Cellisten, und der Situation im MZH nun unsere nächsten Wochenenden einnahm. Dabei lernten wir vor allem, dass es auf richtigen Timing ankommt, nicht nur beim Stück selber, sondern auch in den Vorbereitungen (Aufbau, Requisitenbeschaffung, Fertigstellung einzelner Technikfeatures). Aber wir wurden ein Team, immer wieder von Jörg beschworen mit einem Hinweis auf italienischen Fußball.
Bei den Proben selber kam es immer wieder zu leichten Veränderungen im Skript, das aber von

Anfang an als gut ausgearbeiteter Entwurf galt. Einzelskripte für jede Komponente entwarf sich jede Untergruppe selber, was vor allem für die Licht- und Tonsteuerung sehr wichtig war.

9.5 Werbung: Programmheft und Plakat

Neben diesen Vorbereitungen auf die Projektpräsentation kam auch die Idee auf, dass zu einer Theateraufführung noch mehr gehört. So wollten wir ja schon immer nicht nur vor unserem Fachbereich auftreten sondern etwas Öffentlichkeit haben. Dazu braucht man Werbung, ebenso wollten wir noch ein Programmheft entwerfen, dass nicht nur die Mitspielenden aufführt, sondern auch noch einmal die wichtigen Texte und weitere Zitate zum behandelten Thema enthält. Darüberhinaus sollte es für das Publikum (und natürlich auch für uns) etwas Bleibendes auch nach der Aufführung sein.
Wir begannen mit dem Programmheft, in der Hoffnung mit dem gewonnenen Material auch ein Plakat oder ähnliches (z.B. Flyer) zu gestalten zu können. Für das Heft hatten wir als Ausgangsmaterial Fotos von den Proben, die Matthias gemacht hatte, und einige Zitate, die Bernd aus verschiedenen Quellen herausgesucht hatte, sowie weiteres Bildmaterial. Aus diesen Teilen suchten nun einige von uns Bilder und Zitate aus, von denen wir dachten, dass sie einen guten Eindruck vom Stück geben konnten. Matthias übernahm das Layout, Torsten organisierte den Druck der Hefte und heraus kam das Begleitheft zum Theater der Maschinen, das beim Projekttag, sowie in Stuttgart verteilt wurde.
Mit dem Plakat war es etwas schwieriger, es sollte eine Collage werden, ähnlich wie das Heft. Dies hörte sich einfach und schnell handhabbar an, erwies sich aber im Sammeln und Layout komplizierter als gedacht. So entstand die Idee, aus wenigen Motiven, eigentlich nur der Marionette, der Hauptfigur des Stückes, und einem Zitat von George Dyson ein Motiv zu entwerfen. Daraus wurde das Ankündigungsplakat, das im MZH ausgehängt wurde, sowie einige Flyer mit dem gleichen Motiv, die in der Universität verteilt wurden.

9.6 Die Projekttagpräsentation

Der große Tag der Projektpräsentation stand nun bevor. Wir hatten uns den ersten Termin am frühen Morgen organisiert, so dass wir den Aufbau schon am Abend des Vortages hinter uns bringen konnten, sowie die letzte Generalprobe. Da die anderen Projekte dies auch am Vortrag proben wollten, kam es doch zu kleinen Terminproblemen, bei denen wir uns aber durchsetzen konnten, es wurde nur etwas später am Abend, diese Proben klappten auch noch nicht vollständig. Also wurde die Genaralprobe auf Freitagmorgen direkt vor der Präsentation gelegt. Dies wurde auch der erste Durchlauf des Gesamtstückes ohne große Probleme.
Zunächst musste der Einlass organisiert werden, da die Leute nicht einfach in den Senatssaal hineingehen wie sonst bei einem Projekttag, sondern ab einem bestimmten Zeitpunkt erst die Türen geöffnet werden, weil all dies auch schon zur Aufführung gehörte, wie auch das Abspielen eines Botho Strauß Zitates exakt siebenmal hintereinander (wobei einige nach dem vierten Mal den Saal verließen). Die Aufführung selbst verlief ohne Probleme - so wie es sein sollte.
Die Reaktion des Publikums war gröstenteils positiv, weil wir im Gegensatz zur übliche Präsentation der Projekte einfach unser Ergebnis präsentieren konnten, allein das hob uns von allen anderen (wie auch voriger Projekte) ab. Inwiefern das Publikum eine künstlerische Aussage gesehen hatte und worin die hätte liegen sollen, soll hier nicht weiter erörtert werden.
Zur Aufführung selber kam noch der Präsentationsstand während des Projekttages, den wir etwas spät aufbauten, weil dort einige unserer Bühnenobjekte zu sehen waren, die erst abgebaut werden mussten, was aber insgesamt in Rekordzeit klappte. Dort erklärten wir auch interessierten Menschen die technischen und informatischen Hintergründe des Projektes, die bei der Aufführung selbst nicht zu sehen waren, aber bei einem Informatikprojekt sehr wichtige Be-

standteile sind und gerade die Menschen aus dem Studiengang interessieren.

9.7 Die Aufführung in Stuttgart

Den Abschluss des Projektes bildete eine öffentliche Aufführung, da wir von Anfang an unser Stück nicht nur im Fachbereich vorstellen wollten.
Im Rahmen des Theatertreffens **Schöne Aussicht** organisierte Jörg Richard ein Arbeitstreffen **NETKIDS und THEATER - Zeit ohne Generationen?**. Wir hatten dort die Möglichkeit vor einem am Thema interessierten Publikum unser Stück aufzuführen und auch deren Reaktion zu erleben.
So fuhren wir also auf Einladung dieses Theatertreffens für drei Tage nach Stuttgart, um uns zu präsentieren. Am Montag die Anreise mit der Bahn und mit Auto (=Transporter) für das gesamte Equipment. Am Montagabend bauten wir unsere provisorische Bühne auf einer richtigen Theaterbühne auf, was insbesondere für die jetzt professionelle Lichtsteuerung ungewohnt war und neue Probleme wie auch mehr Möglichkeiten mit sich brachte. Am Dienstagmorgen probten wir zwei weitere Male, am Nachmittag nun DIE Premiere vor echtem Publikum.
Diese Aufführung lief nicht ohne Komplikationen ab, das FliWaTüt blieb stehen, die Figur stolperte auf die Bühne, aber im großen und ganzen hat es funktioniert (Italien hat auch im Finale verloren). Das Publikum hier war nun weniger technikinteressiert, so dass es auch kaum die Gelegenheit nutzte, unsere Objekte aus der Nähe zu betrachten und sich den technischen Hintergrund erklären zu lassen. Im Abschlussgespräch dieser Gruppe spielte unsere Aufführung ebensowenig eine Rolle, aber wir hatten es geschafft, unsere Tournee war beendet.
Am Abend bauten wir alles ab und wieder in das Auto, feierten ein wenig und fuhren am Mittwoch wieder zurück nach Bremen.

Kapitel 10

Fazits

10.1 Gesamtfazit

4 Semester sind nun vergangen, seit wir mit dem Projekt begonnen haben, es ist nun auch unser Projekt geworden und, wir haben es geschafft. Aber nicht nur geschafft, sondern wir haben auch etwas geschaffen, nämlich unser eigenes Theaterstück mit unseren Maschinen. Dies ist das wichtigste Ergenis des Projektes. Hierbei ist insbesondere die Außenwirkung, die wir durch die Aufführungen beim Projekttag und insbesondere außerhalb der Universität in Stutgart erreicht haben, hervorzuheben. Es war von Anfang an geplant, nicht nur etwas für uns zu machen, sondern damit auch an die Offentlichkeit zu gehen, was wir damit auch erreicht haben.
Ein weiterer wichtiger Bestandteil des Projektes war die Interdisziplinarität, die wir durch die Zusammenarbeit mit den Kulturwissenschaftlern bzw. mit quasi Außenstehenden wie Marc hatten. Von ihnen haben wir nicht nur viel gelernt, sondern auch gerade weil sie von einer gänzlich anderen Richtung als der Informatik kamen, auch viel über unser Produkt nachdenken mussten, wenn wir beispielsweise die Möglichkeiten und Grenzen einzelner Maschinen erläutern mussten. Insgesamt muss man aber über die Kulturwissenschaftler sagen, dass wir es nicht ohne sie geschafft hätten. Nicht nur dass sie uns inhaltlich unterstützten (ein extra Dank für die Ideen und das Drehbuch) oder den Kontakt zu unserem Cellisten Marc herstellten, sondern auch dafür, dass in der letzten Phase die Organisation ebenfalls noch übernommen wurde, indem ein Probenplan erstellt wurde und auch dafür gesorgt wurde, dass dessen Termine eingehalten werden. In dieser Zeit haben wir sehr viel der Theaterdisziplin mitbekommen.
Die interne Organisation lief dabei nicht immer so gut. Es war dabei nur ein kleines Missgeschick, dass Willi im zweiten Semester ein Forschungssemester (endlich) hatte und somit nicht zur Verfügung stand. Es war vielmehr das Experiment mit der Selbstorganisation, auf das wir uns aber alle auch zum Teil bewusst entschieden hatte, als wir genau dieses Projekt wählten (ohne aber wohl genau zu wissen, worauf wir uns wirklich einlassen würden). Wir hatten immer sehr viele Freiheiten und das war gut so. Wir haben sie aber nicht immer genutzt, sondern eher ausgenutzt, und das war weniger gut. Wir hätten immer fragen können und hätten auch immer eine Antwort bekommen bzw. wären an jemanden verwiesen worden, der eine Antwort kennt, haben dies aber zu wenig genutzt. Ansprechpersonen gab es gerade im artec genug, und sie haben auch immer gerne geholfen, wie beispielsweise Hermann, der extra für uns einen Pneumatikkurs abgehalten hatte. Durch solche Aktivitäten und Kontakte haben wir viel gelernt, auf jeden Fall mehr als in den begleitenden Veranstaltungen, die für unser Interesse meist zu untechnisch aber dafür zu kulturtheoretisch waren.
Zur Unterstützung gerade des artec muss gesagt werden, dass diese sehr gut war, also nicht nur der Personen, sondern auch materiell, weil oft die Dinge, die wir wirklich brauchten entweder auf Kosten des artec erstanden wurden oder persönlich von Willi bezahlt wurden. Auch dafür schulden wir ihm Dank. Die Unterstützung, gerade was die (rechner)ausstattung anbelangt, durch

den Fachbereich war dagegen mies. Wenn hier schon die Unterstützung angesprochen wurde, so müssen wir hier auch mal die Haustechnik erwähnen, mit der wir Dank unserer Wochenendaktivitäten im MZH sehr viel Kontakt hatten. Zu Beginn lief alles etwas schwierig, aber als wir uns aneinander gewöhnt hatten und wussten, wie wir miteinander umgehen müssen, lief alles recht gut ab.
Insgesamt war das Projekt eine gute Erfahrung, wenn auch nicht immer alles so gelaufen ist, wie es hätte sein sollen, und das Endprodukt über vieles hinwegtröstet, aber gerade das macht die Erfahrung Projektarbeit ja auch aus.
Dass man nun auf unseren Ergebnissen aufbauen könnte und mehr Maschinen und/oder längere Stücke entwerfen kann, macht das Projekt natürlich prinzipiell fortsetzbar. Es ist nur die Frage, in welchem Rahmen dies geschehen könnte. Eine weitere Anfrage scheint es ja zu geben, aber wer dann was mit wem dazu organisisert, wird wohl sehr schwierig, was die personellen Resourcen angeht. Am Spaß hingegen, noch mehr in die Richtung zu machen, wird es aber nie scheitern.

10.2 Persönliche Fazits

Fazit von Bernd Robben

2 Jahre Projekt "Theater der Maschinen"!

Freitagmittag klopfen zwei Studenten an und bitten mich zwischen Tür und Angel, doch aus meiner Sicht ein Fazit über das Projekt "Theater der Maschinen" für den Abschlussbericht zu schreiben- bis Montag. Dann soll er gedruckt werden. Symptomatisch für ein Projekt, in dem viel spontanes Engagement gefordert war! Aber natürlich werde ich auch in der kurzen Frist etwas schreiben! Denn es hat mir großen Spaß gemacht, als wissenschaftlicher Mitarbeiter teilzunehmen, die Begleitung ein Semester lang sogar selbständig ohne Willi Bruns als betreuenden Hochschullehrer zu übernehmen, was ich allerdings auch nicht immer einfach fand. Der Aufführung des lange geprobten Stücks am Projekttag fieberte ich genauso entgegen, wie die studentischen Mitglieder. Die befürchteten Pannen blieben aus. Es war ein voller Erfolg. Das ist die erste Bilanz.

Aber das kann wohl nicht das einzige Fazit sein.

Gleichzeitig ungeheuer kompliziert und unglaublich befruchtend war das Durchhalten der Interdisziplinarität im Projekt, die mir ein besonderes Anliegen ist. Ich glaube, dass am Schluss alle Projektmitglieder überzeugt waren: Ohne die Zusammenarbeit mit den Kulturwissenschaftlern hätte es kein Theater der Maschinen als wirkliche Aufführung gegeben.

Schade fand ich, dass eine Reihe von engagierten Teilnehmern zum Teil noch nach dem dritten Projektsemester den Versuchungen des boomenden Arbeitsmarktes für Informatiker erlagen und das Projekt verließen.

Da waren auch viele andere Schattenseiten: Chaos, Durchhänger, manchmal unerträgliche lähmende Unverbindlichkeit, gegen die ich mich aus heutiger Sicht als Betreuer mehr und vor allem konsequenter zur Wehr setzen würde. Jedoch mag an der Formulierung schon klar werden: Ich möchte auf keinen Fall der Versuchung erliegen, die Verantwortung für das Projekt an mich zu ziehen und den Teilnehmern abzunehmen. Die Übernahme dieser Verantwortung halte ich für das wichtigste Lernziel des Projektstudiums.

Ansonsten sind mir im Nachhinein die Freizeitaktivitäten des Projekts auf Helgoland oder in Berlin mehr im Gedächtnis als manche ausufernde Diskussion auf einem Plenum.

Bernd Robben

Fazit von Daniel Mutis

Trotz des anfänglichen "Chaos" kam das Projekt am Ende doch zu einem versöhnlichen Abschluss. Dies lag zum größten Teil an der Mitarbeit der Kulturwissenschaftler in Person von Jörg Richard und Martin, ohne deren mit Hilfe das Projekt bestimmt nicht so gelungen wäre. Sie vermochten es gekonnt die unabhängigen Teilprojekt zu einer Aufführung zusammenzuführen. Deshalb an dieser Stelle noch mal ein Dankeschön an diese beiden.

Gestört hat mich anfangs die Planlosigkeit unseres Vorgehens. So haben sich zwar nach unserer ersten Projektfahrt relativ schnell verschiedene Teilprojekte herauskristallisiert, die recht unabhängig voneinander fungierten. Das Gesamtbild des Ganzen wurde jedoch erst kurz vor Abschluss des Projektes durch die Zusammenarbeit mit den Kulturwissenschaftlern sichtbar.

Recht positiv herauszuheben ist der Führungsstil von Willi Bruns, der uns recht viele Freiheiten

gelassen hat. Es war kein fest vorgegebenes Ziel definiert, sondern wir konnten selbst bestimmen, wohin wir mit dem Projekt steuern. So waren wir zu mehr Eigeninitiative gefordert was jedoch auch teilweise in Lethargie ausartete. Bei zu großen Lethargieanfällen unsererseits hat Willi dann auch mal regulierend eingegriffen.

Der Lerneffekt des Projektes war, zumindest von meiner Seite aus, immens. So habe ich viele neue Sachen hinzugelernt die ich mir selbst aneignen musste. Auch die Zusammenarbeit in Gruppen wurde gefördert, was bei einer späteren beruflichen Tätigkeit im Informationsverarbeitenden Sektor bestimmt hilfreich ist.

Ein kleiner Kritikpunkt muss jedoch bei den begleitenden Lehrveranstaltungen gelassen werden. Diese waren von der Thematik zwar interessant, haben uns bei der Realisierung des Projektes jedoch recht wenig geholfen. Die Ausnahme stellt hierbei der Pneumatik-Kurs da, obwohl wir letztendlich auch keinerlei pneumatische Bestandteile in unserem Projekt hatten. Dies lag aber wohl auch an den mangelnden finanziellen Möglichkeiten eines studentischen Projektes.

Im Grossen und Ganzen würde ich das Projekt von meinem Standpunkt als gelungen bezeichnen, da ich eine Menge gelernt habe, sowohl was die angeeignete Materie als auch das Zusammenarbeiten in Gruppen betrifft. Auch das Endprodukt war trotz der aufgetretenen Probleme recht sehenswert.

Fazit von Denis Böhme

Nach einer anfänglichen Euphorie, die sich im Projekt breit machte, tat sich eine Zeit lang nicht viel. Die Teilnehmerzahl halbierte sich bis zum Ende des Projekts. Ein gemeinsames Ziel war in der Mitte des Projekts nicht zu erkennen.

Eine weitere wichtige Erfahrung, war die Erkenntnis, das man sich auf Niemanden mehr verlassen wollte. Es war oft einfacher bestimmte Schritte alleine durchzuführen, als sich von Anderen abhängig zu machen. Die Ursache darin lag selten an Faulheit, sondern in den verschiedenen Ansprüchen der Teilnehmer. Unsere auf zwei Leute- von Anfangs Vier- zusammengeschrumpfte Teilgruppe, erwies sich am Ende dabei noch als funktionierend.

Die Unterstützung vom Fachbereich 3 war extrem dürftig, im Gegensatz zu der des Artec. Und das am Ende kein Projektraum mehr zur Verfügung stand, wird sich wohl in der Qualität des Projektberichts bemerkbar machen.

Die Freiheit die uns Willi Bruns bei den Projekt gelassen hatte bekam einigen Leuten und Ihren Produkten sehr gut. Andere hatten mit diesen extrem selbstständigen Arbeiten arge Probleme.

Gegen Ende des Projekts nahm das Theater wieder Fahrt auf, dank der Intervention der Kulturwissenschaftler. Unter Jörg Richard änderte sich der Führungsstil, es musste gemacht werden, was gesagt wurde. Jetzt konnten Leute Ihre Qualitäten beweisen, die zuvor mit den sehr selbstständigen Arbeiten nicht so richtig zurecht gekommen sind.

Überraschend war das Ergebnis, welches am Ende des Projekts vorgeführt werden konnte. Wenn die vertüdelten zwei Semester produktiver genutzt worden wären, so hätte man ein unschlagbares Ergebnis erzielen können.

Das klingt alles etwas trüb, aber das Interessante ist, das unser Projekt wohl nur schwer zu überbieten sein wird. Ganz abgesehen von den legendären Projektfahrten.

'Was hat das mit Informatik zu tun?', war der Wortlaut der Frage eines Mannes nach der Vorführung. Auf eine dumme Frage bekam der gute Mensch eine angemessene Antwort. Aber

wenn wir tatsächlich ein Projekt gemacht hätten, was nix mit Informatik zu tun hätte, so wäre es nicht automatisch weniger Wert gewesen.

Fazit von Ersin Ürer

Die Idee des Projektes als wesentlichen Bestandteil des Studiums finde ich sehr positiv.

Am Anfang des Projektes haben wir Referate gehalten, um mehr Informationen über das Thema zu sammeln. Außerdem haben mir die parallel laufenden Veranstaltungen sehr gefallen. Innerhalb des Plenums war alles gut organisiert. Die Projektfahrten haben uns geholfen, uns in kürzester Zeit kennenzulernen, so dass die Zusammenarbeit innerhalb der Gruppe gut war.

Die Vorbereitungen und die Ideen für die Präsentation des Projektes haben Spaß gemacht. Gut fand ich daran, dass dabei alle Mitglieder des Projektes aktiv zusammengearbeitet haben.

Positiv fand ich auch, dass uns Willi selbstständig arbeiten ließ, und sich nur eingeschaltet hat, wenn es nötig war. Dadurch hatte man das Gefühl selbst für seine Arbeit verantwortlich zu sein. Nach Abschluss dieses Projektes kann ich sagen, dass sich die Arbeit dafür sehr gelohnt hat.

Fazit Frank Euhus

Tja, was habe ich in diesen 4 Semestern Projektarbeit gelernt?
Zunächst einmal habe ich nicht ohne Bedacht ein selbstorganisiertes Projekt gewählt, das nicht von vornherein die zu erwartenden Ergebnisse vorwegnahm. In dieser eigenen Organisation des Projektes habe ich mich dann auch ziemlich engagiert, weil ich darin schon einige Erfahrungen habe und mir so etwas auch Spaß bringt. Trotzdem, oder gerade deswegen habe ich mich auch manchmal aus diesen Dingen herausgehalten, um dort nicht alles an mich zu ziehen. Ich denke, dieser Spagat hat recht gut geklappt.
Die Gruppe war von Anfang an sehr groß für ein Projekt, recht unüberschaubar und mir nicht vollständig bekannt, da ich ein Quereinsteiger im Informatikstudium bin und nicht alles so mitgemacht habe wie die meisten. In diese bunte Truppe, die zu einem grossen Teil aus solch *unüblichen* Informatikstudis bestand, passte ich aber rein. Viele hatten schon vorher etwas anderes als Ausbildung gemacht oder arbeiteten schon viel; dies führte leider auch zur hohen Abbrecherquote, was ich sehr bedauerlich fand.
Zurück zum Gelernten: Ein Projekt ist langatmig, mit Höhen und Tiefen und einigen Durchhängern, aber wenn alle mitmachen, kann es klappen. Das war mir vorher an sich klar, aber selbst in einem Zweijahresprojekt hatte ich dies noch nicht erlebt, welch eine Erfahrung. Die Menschen sind sehr verschieden mit unterschiedlichen Qualitäten, die sich manchmal ganz anders offenbaren, als man vorher dachte; auch so etwas ahnte ich, ohne diese Erfahrung gemacht zu haben. Soweit zu den Erwartungen.
Zum Neuen: Das Wunder, aus einer Sackgasse(?) heraus noch alles zu meistern und das auch unter Termindruck war schon beeindruckend, wenn ich mir das Resultat des Projektes ansehe. Die große Erfahrung, endlich mal auf einer Bühne gestanden und agiert zu haben, super. Die Zusammenarbeit mit den KuWis, die eine vollkommen andere Herangehensweise an unsere Probleme hatten. All die Menschen, die ich im Laufe des Projektes kennenlernen (im wahrsten Sinne) durfte. Und last, but not least, die Erkenntnis, dass ich niemals ein richtiger Informatikstudent werde.
Aber, wie alle Kenner des DerDuDie-ismus wissen: Jeder lebt in seiner eigenen Welt, aber meine ist die richtige, denn die Wirklichkeit sieht in echt ganz anders aus.

Fazit von Jörn Raffel

Für mich waren die 2 Jahre, die das Projekt gedauert hat, sehr lehrreich. Gerade die erste Zeit der Ideenfindung war zwar teilweise ziemlich frustrierend, aber ich denke, dass das meiner Meinung nach sehr gute Ergebniss diese Phase rechtfertigt.

Willy und Bernd haben uns viel Freiraum gegeben, aber auch darauf aufmerksam gemacht, wenn Sachen nicht so liefen wie sie sollten.

Ausserdem war die materielle Unterstützung durch das Artec im Gegenssatz zum Fachbereich 3 sehr gut.

Die Zusammenarbeit mit den Kulturwissenschaftlern Joerg Richard und Martin Phillip hat mich beeindruckt, besonders bemerkenswert fand ich deren Geduld mit uns, bzw. mit unserer mangelnden Disziplin. Ohne die beiden wäre das Projekt bzw. die Aufführung lange nicht so erfolgreich geworden, sie haben einen sehr grossen Teil dazu beigetragen.

Schade finde ich, dass die sich die Anzahl der Projektmitglieder während des Projektes halbiert hat. Umso erstaunlicher ist es aber, dass wir trotzdem im Grossen und Ganzen unser Ziel erreicht haben, bzw. es durch die Hilfe von Jörg Richard und Martin Phillip teilweise noch überschritten haben.

Fazit von Khaled Abdelrahimsai-Pjau

Wenn man unser Projekt vom Ergebnis her mit den anderen Projekten (Jahr 2000) vergleicht, dann kommt man eindeutig zur einer positiven Bilanz. Es war in diesem Jahr mit Abstand das beste Projekt. Hinter so einer hervorragenden Vorführung steckte die Arbeit von Studenten und Betreuern, die trotz menschlicher und technischer Hindernisse geschafft haben einen gemeinsamen Nenner zu finden, um das Ziel Erfolgreich zu erreichen. In den zwei Jahren gab es einige Zweifel, ob wir überhaupt etwas vorführen könnten, da die Zusammenarbeit unter einigen Unterprojekten sehr unbefriedigend war. Quereinsteiger, Absprünge und Gleichgültigkeit von Stundenden machten dies noch schwieriger. Aber der Glaube an das eigene Können und das Arrangement überwindeten die Hindernisse und somit stärkte dies das Vorankommen des gesamten Projektes. Ich bin der festen Meinung, dass unser Projekt ohne Mithilfe von den Kulturwissenschaftler nicht so erfolgreich geworden wäre. Die Kulturwissenschaftler haben nicht nur Farbe in unserem Projekt gebracht, sondern dank denen gab es eine gewisse Disziplin in der Arbeitsweise unter den einzelnen Unterprojekten. Sehr interessant und bemerkenswert fand ich die Zurückhaltung von den Betreuern (Willi Bruns und Bernd Robben). Obwohl Willi Bruns den Anschein hatte, er würde nicht viel wissen, was in den einzelnen Unterprojekten geschehen würde, war er bemerkenswert sehr gut informiert über den Ablauf und Geschehen in dem Projekt. Dies hat sicherlich mit seinen jahrelangen Erfahrungen mit den studentischen Projekten zu tun. Das Projekt *Theater der Maschinen* war eine sehr wertvolle Bereicherung hinsichtlich Gruppenarbeit, Informatikwissen und deren Randgebieten. So ein Projekt ist eine Vorbereitung auf das spätere Arbeitsleben, den im realen Leben muß man enormen Wissen bezüglich Gruppenarbeit mitbringen und vorweisen können.

Fazit von Marcus Fährer

“Theater der Maschinen”, oder “So sollte ein studentisches Projekt aussehen”

Als ich vor zwei Jahren zum ersten Treffen des Projektes “Theater der Maschinen” ging, hatte ich natürlich eine Vorstellung davon, wie für mich ein studentisches Projekt ablaufen sollte. Für mich sah ein Projekt so aus, daß die Studenten ein sehr großes Mitspracherecht haben, und alles gemeinsam beschlossen wird, und nicht daß der Professor von oben herab alles diktiert, was im

Projekt geschehen soll. Jetzt im Nachhinein muß ich sagen, daß meine Erwartungen bei weitem übertroffen worden sind. Wir "durften" nicht nur Mitentscheiden, sondern wir "mußten" fast alles selber entscheiden. Dieses Vorgehen war für uns absolut super, da wir so alle Phasen eines Projektes hautnah miterleben konnten. Diese absolute Selbstbestimmung hatte aber natürlich nicht nur Vorteile, sondern brachte auch einige Probleme mit sich, die wir dann auch selber lösen "durften".

Die Motivation innerhalb des Projektes war sehr unterschiedlich. Zunächst waren wir alle sehr motiviert, doch dies änderte sich nach einiger Zeit, als wir uns nicht auf anhieb auf eine Idee für unser Projekt einigen konnten. In dieser Ideenfindungsphase ging die Motivationskurve etwas bergab, was sich aber nach unserer ersten Projektfahrt änderte auf der wir eine gemeinsame Idee für unser Projekt entwickelten. Das zweite Motivationstief hatten wir im dritten Semester, als Willi sein Urlaubssemester hatte. Ob diese zwei Gegebenheiten irgendwie zusammenhingen, kann ich nicht genau sagen, aber auf jeden Fall fand ich den Zeitpunkt für ein Urlaubssemester nicht sehr glücklich gewählt. Kurz darauf ging die Motivationskurve (zumindestens bei mir) aber wieder steil nach oben, als wir anfingen mit den Kulturwissenschaftlern unser "Theaterstück" zu proben.

Alles in allem empfinde ich das Projekt als zu 100% gelungen, da wir alle Vor- und Nachteile eines Projektes dieser Grössenordnung erlebt haben. Auch die weniger guten Erfahrungen sehe ich jetzt im Nachinein als positiv, da solche Probleme eben bei solchen Projekten auftreten, und dann halt möglichst gut gelöst werden müssen. Ich habe zu keinem Zeitpunkt meine Entscheidung für dieses Projekt bereut, und würde es jetzt im Nachhinein auch sofort wieder wählen.

Marcus Fährer

Fazit von Matthias Liebert

Das viersemestrige Projekt des Hauptstudiums ist wohl eines der einschneidensten Erlebnisse im Bremer Informatikstudium. Sicherlich der Zeitraum im Studium in dem man etwas größere Dinge angehen und bewegen kann. Nach einer anfänglichen schleppenden Phase der Projektzielausarbeitung, die durch mehrere Verwerfungen der Ideen gekennzeichnet war, in ihrer Dauer jedoch gerechtfertigt gewesen ist, tauchte das Projekt scheinbar in die Phase des Stillstandes ein. Den wirklich lobenswerten antiautoritären Führungsstil der Projektveranstaltenden machten sich leider ein Teil der Projektteilnehmer zu Nutze, indem sie in physische und psychische Regungslosigkeit verfielen; wohl nicht erkennend, dass Freiheiten auch mit Verbindlichkeiten, wie z.B. Eigenverantwortung, eng miteinander verknüpft sind. Aus dieser Phase halfen uns Jörg Richard und Martin Phillip heraus, die einen theaterlichen Rahmen für unsere Idee entwickelten. Die uns noch verbleibende Zeit, immerhin noch ein ganzes Semester, nutzten wir dann doch zum ersten Mal in diesem Projekt sehr intensiv, um unsere Unterprojekte den Erfordernissen des Theaterstückes anzupassen. Dabei erweckten einige Projektteilnehmer den Eindruck, dass sie erst in der letzen Phase des Projekts damit begannen ihr Unterprojekt zu entwickeln; natürlich mit den entsprechenden Resultaten.

Wie auch immer, den Abschluss des Projektes mit den Aufführungen auf dem Projekttag der Informatik und auf dem Theaterfestival "Schöne Aussicht" kann man als wirklichen Erfolg werten.

Es ist nur schade, dass über den Zeitraum von vier Semester wirklich nur einige Leute die Gelegenheit wahrgenommen haben, sich derart für das Projekt zu engagieren, um es maßgeblich mitzugestalten. Von diesen Leuten haben leider auch noch ein paar das Projekt aus beruflichen Gründen frühzeitig verlassen. Das Verhalten gegenüber unseren engagierten "Gästen", die uns

letztlich freiwillig und ihre Freizeit opfernd halfen, war in manchen Situationen sehr beschämend. Ich kann für meine Person nur festhalten, das ich neben der Projekterfahrung viel Fach- und Menschenkenntnisse erworben habe. Sehr faszinierend empfand ich die interdisziplinäre Zusammenarbeit, die mir persönlich eine Menge neuer Eindrücke auch über das Studium hinaus erbrachte. Es ist nur bedauerlich, das dieses Projekt wesentliches "Größeres" hätte leisten können mit ein wenig mehr an Engagement.

Fazit Oliver Weickmann

Der Beginn unserer Gruppenarbeit zeichnete sich, angetrieben von der allgemeinen Gruppendynamik, durch Ideenreichtum aus. Da wir allerdings in Hinblick auf die vermeintlich lange Zeit von vier Semestern entstehende Probleme eher vor uns her schoben, als diese zu lösen, verbrachten wir sehr lange Zeit damit Probleme zu diskutieren. Wir erkannten leider erst zu spät, dass die ursprünglich geplante Maschine nicht in dem Umfang realisierbar war, wie wir uns das vorgestellt hatten. Da wir zu diesem Zeitpunkt aber immer noch nicht unter Zeitdruck standen, haben wir uns auch nach der Suche von Alternativen zu viel Zeit gelassen. Erschwerend kam hierbei noch dazu, dass erst sehr spät ein gültiges Gesamtkonzept für unsere Aufführung erstellt wurde. Da wir unsere Maschine in dieses Konzept einbinden wollten, sollte sie in diesem Rahmen auch eine Funktion übernehmen können.
Praktisch erst in ,,letzter Minute" wurde das Konzept des Mülleimers geboren. Da wir nur zwei Gruppenmitglieder waren, reichte daher unsere verbleibende Zeit nicht zur kompletten Realisierung des Roboters aus. Trotzdem gestalteten wir eine Maschine die sehr gut in die Theateraufführung integriert werden konnte. Leider blieb hierbei gerade der Teil auf der Strecke, der für ein Projekt der Informatik eher als wichtig eingestuft werden könnte : das Erstellen einer Steuerung mit dem Computer. Durch das Fehlen dieser Komponente war es dann leider nicht möglich eine informationstechnische Anbindung an das Projekt zu gestalten. Dennoch bin ich der Meinung, dass wir durch den Bau der Maschine und der Integration in die Aufführung einen wesentlichen Teil zum Projekt beigesteuert haben.
Ich denke durch eine strukturiertere und besser geplante Arbeit hätten wir mehr ereichen können, trotzdem ist das Erreichte sehenswert.

Die Anfangs- und Endphase des Projekts war meiner Meinung nach die produktivsten. Der Beginn des Projekts war durch eine Vielzahl guter Ideen und Diskussionen geprägt. Die Bereitschaft der einzelnen Projektmitglieder ihren Teil zum Projekt beizutragen war überwiegend gut. Sehr gut fand ich dass nach der Unterteilung in einzelne Untergruppen, diese durch regelmäßige Berichte Probleme und Fortschritte den anderen Mitgliedern mitteilten. Es wurde gerade in der Endphase sehr viel Energie und Ideenreichtum aufgewendet um die entstandenen Probleme zu lösen. Gerade die Improvisation, die zu einer gelungenen Aufführung beitrug, empfand ich als sehr angenehm. Die letzten Wochen vor dem Projekttag waren für mich die angenehmsten, weil hier der Teamgeist und die Bereitschaft der Einzelnen zur Teilnahme am höchsten war.
Leider war ab dem zweiten Semester die Bereitschaft zum sogenannten Plenum zu erscheinen eher gering. Dadurch wurden auch die regelmäßig geführten Referate oder Berichte ad absurdum geführt. Meiner Meinung nach wurde die Projektarbeit im zweiten und dritten Semester von vielen Teilnehmern sehr stark vernachlässigt. Erst im vierten Semester wurde wieder intensiv am gemeinsamen Ziel gearbeitet. Da allerdings keine detaillierten Berichte oder Referate über bestehende Fortschritte gehalten wurden, fand ich es sehr schwierig die Arbeiten der Anderen zu verstehen oder zu bewerten. Im nachhinein denke ich eine straffere Projektleitung hätte zu einem besseren Resultat geführt. Die Tatsache, dass wir freie Hand hatten in der Durchführung des Projekt führte oftmals dazu dass Vieles erst spät oder gar nicht getan wurde. Ein Plan mit festen Eckdaten an denen bestimmte Dinge fertiggestellt sein müssen, hätte schon von Anfang an bestehen müssen, und die Einhaltung dieses Planes wäre meiner Meinung nach die Aufga-

be der Projektleitung gewesen. Das völlige Zurückziehen der Leitung des Projektes war eher hinderlich für das Projekt. Dies führte allerdings zu meiner Erkenntnis bei einer vergleichbaren Projektarbeit eine bessere Planung zu erstellen. Wenn ich ein weiteres Mal ein Projekt wählen würde, müsste dies eine andere Art der Projektleitung haben.
Ich war sehr über das Ergebnis unserer Arbeit überrascht. Die Aufführung war sehr gut und daher empfinde ich unser Projekt auch als vollen Erfolg.

Oliver Weckmann

Fazit von Thomas Gnewuch

Ein Projekt zu veranstalten innerhalb des Hauptstudiums halte ich für eine sehr gute Idee. Auch die Dauer von etwa 2 Jahren ist in Ordnung. Hier haben wir die Möglichkeit gehabt, mal zu schauen wie diese Projektarbeit an sich abläuft, welche Probleme und Möglichkeiten sich bei einer solchen Zusammenarbeit ergeben. Prädikat wertvoll ! Ich denke auch im späteren Berufsleben ist eine solche Art der Zusammenarbeit eher der Standard als die Ausnahmen. Heutzutage entwickelt niemand (fast niemand) mehr Software allein. In diesem Punkt des Lernens der Koordination und der Durchführung habe ich sehr viele Erfahrungen sammeln können. Soviel erstmal zur Existenz eines 4 semestrigen Projektes an der Bremer Uni.

Ansonsten ist das Projekt auch gut gelaufen. Die anfänglichen Schwierigkeiten bei der Ideenfindung haben mich etwas irritiert. Es ist viel Zeit "verloren" gegangen, die aber nicht umsonst war. Viele Veranstaltungen die zum Projekt angeboten wurden waren gut, einige davon hätte man sich aber wirklich schenken können.

Die Zurückhaltung von Willi Bruns war auch wohldosiert. Er hat nur eingegriffen, wenn es kurz vor einem Brand innerhalb des Projektes stand. Ansonsten hat er uns relativ freie Hand gelassen,was sich letztendlich auch als richtig herausgestellt hatte.

Insgesamt muss ich sagen, es hat sehr viel Spass gemacht, ich habe für mich viele wertvolle Erfahrungen mitnehmen können, und würde so ein Projekt jederzeit wieder mitmachen, wenn ich die Zeit dazu hätte.

Thomas Gnewuch

Fazit Torsten Fröhling

Das Projekt THEMA, Theater der Maschinen von oder vielmehr mit Prof. Dr. F. Wilhelm Bruns, ist jetzt beendet. Rückwirkend betrachtet gibt es wohl sehr viele positive Erkenntnisse, Erfahrungen und Momente. Zu Erwähnen wäre da zum Beispiel die freie Führungsarchitektur, die selbst in haarigen Momenten stets beibehalten wurde und auf Dauer gesehen zu einem doch sehr guten Ergebnis geführt hat. Auch die überdurchschnittlich vielen privaten Treffen in Berlin, Helgoland oder an der Nordsee haben in meinen Augen zu einem starken Gruppengefühl und somit zu dem Gesamterfolg geführt.

Als positive Erkenntnisse kann man im nachhinein auch grobe Fehlentscheidungen werten, die man so wohl in seinem Leben nicht noch einmal treffen würde. Mein liebstes Beispiel hierfür wäre die Gruppenwahl am Anfang des Projektes. Unter dem Motto: "das kriegen wir schon hin (1,5 Mann Gruppe)" hatten wir leichtsinnig Lasten auf uns genommen, die wir so nicht leisten konnten. Bei einem zwei jährringen Projekt sollte man solch wichtige Entscheidungen nicht leichtfertig treffen.

Sicherlich gab es auch negative Dinge, wo würden diese wohl ausbleiben, in meinen Augen hätten die Begleitveranstaltungen sich viel mehr auf das eigentliche Projekt beziehen sollen. Sicherlich werden jetzt viele sagen, das die einzelnen Unterprojekte zu verschieden waren, aber in meinen Augen findet man an sehr vielen Stellen gleiche Grundlagen wieder. Besonders möchte ich an dieser Stelle die Workshops "Grundlagen der Pneumatik" hervorheben, meines Erachtens von allen gerne genutzt worden, praktisch etwas zu lernen.

In meinen Augen ist ein zwei jährringes Projekt eine sehr lohnende Sache. Es treten Probleme auf die viele im späterem Berufsleben wiederfinden werden und dann eventuell anders handeln. Man bekommt einfach Eindrucke aus dem Projektleben vermittelt, die man mit einer Lehrveranstaltung oder einem deutlich kürzerem Projekt nicht vermitteln könnte. Der einzige Nachteil ist dabei allerdings der Zeitraum. Das Projekt findet zu einem Zeitpunkt statt, zu dem schon viele irgendwelche Berufliche Verpflichtungen eingegangen sind. Das soll kein Freischein zum Fernbleiben sein, aber bei zwei Angestellen, wie in meinem Fall, hat man auch denen gegenüber die Verantwortung das Geschäft so zu führen, das sie auch in den nächsten Monaten noch einen Arbeitsplatz haben.

Alles in allem muß ich sagen, daß es viel Spaß gemacht hat und das man dabei vieles gelernt hat. Ich habe zu keinem Zeitpunkt an der richtigen Wahl des Projektes gezweifelt. Im nachhinein hätte ich gerne mehr für mein Unterprojekt getan, aber das läßt sich ja nicht mehr ändern.

Torsten Fröhling

Fazit von Vahit Bilmez

Meine persönlichen Reflexionen auf das Projekt in kurzen Worten zusammenzufassen, fällt mir sehr schwer.

Ich habe in diesem Projekt ohne Zweifel viel gelernt, viel Spaß - und auch Streß gehabt. Ich habe meine bisherigen Erfahrungen in Bezug auf Gruppenarbeit, Motivation, Organisation, Terminplanung - und auch Einhaltung , deutlich erweitern können.

Sehr positiv empfand ich auch die Erfahrung, mit Leuten zusammenzuarbeiten, die ich vor dem Projekt noch nicht gekannt hatte. So hat mir das Projekt ebenso in "menschlicher" Hinsicht einiges gegeben, auch in den Auseinandersetzungen, die natürlich auch, und mit überraschend positiven Ergebnissen stattfanden.

Die Planung und Organisationen eines Projektes mit vielen Leuten über zwei Jahre ist eine Aufgabe, mit der sich nichts im übrigen Studium vergleichen läßt. Hier spielen viele gruppendynamische Faktoren eine wichtige Rolle, mit denen man im übrigen Studium üblicherweise nicht konfrontiert wird.

In diesen Hinblick werde ich mit Sicherheit die zahlreichen gemeinsamen Projektwochenenden (Cuxhaven, Helgoland, Berlin, Stuttgart, usw.) in Erinnerung halten.

Alles in Allem: Es hat sehr viel Spaß gemacht, ich habe viel gelernt, wir haben unseren Ziel erreicht und erfolgreich das Projekt zu ende geführt. Ich würde es mit diesen Freunden, in dieser Gruppe, auch ein nächstes Projekt angehen, auch wenn ich dann einiges anders machen und angehen würde. Ich bedanke mich bei allen Freunden, die in diesem Projekt mitgearbeitet haben.

Fazit von Willi Bruns

Studentische Projekte im Hauptstudium verstehe ich als Vorbereitung auf eine konfliktreiche und spannungsvolle betriebliche und wissenschaftliche Praxis.

Für Ingenieure und Informatikerinnen scheint mir neben der formalen Denk- und Handlungsweise, die Schwerpunkt jeder traditionellen Ausbildung in Informatik ist, die Fähigkeit wichtig zu sein, Unsicherheiten auszuhalten, Ambivalenzen wahrzunehmen und zu akzeptieren. Informatik strebt nach Sicherheit und Eindeutigkeit. Lebenswelten und ihre sozialen und persönlichen Aspekte bringen Mehrdeutigkeiten und Zweifel. Was liegt also näher, als die Wissenschaft der automatisierten Abstraktion (Informatik) mit der Inszenierung von Lebenswelt (Theater) zu verbinden. Ich bin stolz auf "meine" Projektgruppe Theater der Maschinen, dass sie diese Brücke in vielfältiger und individuell sehr unterschiedlicher Weise gestaltet und erlebt hat.

Ich bin den Kulturwissenschaftlern Jörg Richard und Martin Philipp dankbar dafür, dass sie sich auf dieses Abenteuer eingelassen haben, es mit so vielen Ideen befruchteten und für eine Verbindlichkeit sorgten, die uns allen gut getan hat.

Besonders bedanken möchte ich mich auch bei unserem Super-Chellisten Marc Froncoux und der wunderbaren Stimme von Ruth Reinecke.

Schade, dass Projekte enden.

Willi Bruns

Literaturverzeichnis

Mülleimer

[1] *Stanley B. Lippmann:* C++ Einführung und Leitfaden; 1995; ISBN 3-89319-375-8

[2] *Andrew S. Tannenbaum:* Computernetzwerke; 1998 ISBN 3-8272-9568-8

[3] Duden "Informatik"; Hrsg. vom Lektorat des BI-Wiss.-Verl. unter der Leitung von Hermann Enhesser; 1993; ISBN 3-411-05232-5

[4] *HTML:* http://www.atmel-wm.com/products/development_tools.php3?dom1=21

[5] *HTML:* http://www.atmel-wm.com/products/documentation.php3?dom1=21

Master Control Program

[6] *Bjarne Stroustrup:* Die C++ Programmiersprache; Addison Wesley Deutschland GmbH, Bonn, 3. Auflage, 1998

[7] *Niklaus Wirth:* Compilerbau; B.G. Teubner, Stuttgart, 1984

[8] *Arno Puder, Kay Römer:* MICO is CORBA; dpunkt Verlag, Heidelberg, 2. Auflage, 1999

[9] *John E. Hopcraft, Jeffrey D. Ullman:* Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie; Addison Wesley Deutschland GmbH, Bonn, 3. Auflage, 1994

Der Soundraum

[10] *Petzold, Charles:* Windows - Systemprogrammierung; Microsoft Press Deutschland, Unterschleißheim, 5. Auflage, 1999

[11] *Bradley Bargen, Peter Donnelly:* Inside DirectX; Microsoft Press Deutschland, Unterschleißheim, 1998

[12] *Bjarne Stroustrup:* Die C++ Programmiersprache; Addison Wesley Deutschland GmbH, Bonn, 3. Auflage, 1998

[13] *Niklaus Wirth:* Compilerbau; B.G. Teubner, Stuttgart, 1984

[14] *Arno Puder, Kay Römer:* MICO is CORBA; dpunkt Verlag, Heidelberg, 2. Auflage, 1999

[15] *John E. Hopcraft, Jeffrey D. Ullman:* Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie; Addison Wesley Deutschland GmbH, Bonn, 3. Auflage, 1994

[16] *David Kurglinski, George Sheperd, Scot Wingo:* Inside Visual C++ 6.0; Microsoft Press Deutschland, Unterschleißheim, 1998

Lichtsteuerung

[17] *Johannes Webers:* Handbuch der Film- und Videotechnik- Die Aufnahme, Speicherung, Bearbeitung und Wiedergabe audiovisueller Programme; Franzis Verlag München, 4.Auflage, 1991;

[18] *Jost J. Marchesi:* Professionelle Beleuchtungstechnik; 3. Auflage, 1994

[19] *Max Keller:* Faszination Licht- Licht auf der Bühne; 2. überarbeitete. u. aktualis. Aufl., 2000

[20] *Roland Baer, Martin Ecker, Dietrich Gall:* Beleuchtungstechnik, Anwendungen; 1993

[21] *Helmut Lindner, Harry Brauer, Constans Lehmann:* Taschenbuch der Elektrotechnik und Elektronik; 7. Auflage, 1999;

[22] *Ulrich Dietmeier:* Formelsammlung für die elektronische Schaltungstechnik; Oldenbourg, München, 1997

Fliwatüt

[23] *PDF, www.atmel.com:* Reference AT90S8515

[24] *PDF, www.atmel.com:* Instruction Set AT90S/L4434 and AT90/LS8535

[25] *PDF, www.atmel.com:* AVR Studio User Guide

[26] *PDF, www.atmel.com:* AVR Assembler User Guide

[27] *PDF, www.atmel.com:* AVR Simulator Manual

[28] *PDF, www.atmel.com:* AVR Development Tools

[29] *PDF, www.atmel.com:* AVR Studio User Guide

[30] *PDF, E-Lab Digital Engineering:* EDE1204 Bi-Polar Stepper Motor IC

[31] *PDF, Texas Instruments:* L293 Quadruple Half-H Driver

[32] *HTML, www.atmel.com:* Control of Stepping Motors, a tutorial

[33] *Schwarz, Stefan / Potucek, Rudolf:* Das TEXikon

[34] *Turianskyj, Nino / Heimann, Frank:* Visual C++ 6

[35] *Kruglinski, David / Sheperd, George / Wingo, Scot:* Inside Visual C++ 6.0

Marion

[36] *HTML, www.gtk.org:* Gtk+ - The Gimp Toolkit

[37] *HTML, www.atmel.com/atmel/products/prod23.htm:* AVR 8-Bit RISC

[38] *HTML, http://glade.gnome.org:* Glade Gtk+ user interface builder

[39] *HTML, www.qcad.org:* QCad Homepage

[40] *HTML, www.rtlinux.com:* RTLinux FSMLabs Homepage

[41] *HTML, www.easysw.com/ mike/serial/serial.html:* Serial Programming Guide for POSIX Operating Systems

[42] *HTML, /usr/doc/HOWTO/mini IO-Port-Programming:* Linux IO-Port-Programming HOWTO

[43] *HTML, www.beyondlogic.org/spp/parallel.htm:* Interfacing the Standard Parallel Port

[44] *HTML, www.linux.org:* Linux

Von der Wurst zum Kubismus

[45] *Hans-Lothar Hase:* Dynamische virtuelle Welten mit VRML 2.0: Einführung, Programme und Referenzen; Heidelberg: dpunkt- Verlag für digitale Technologie, 1997; ISBN 3-920993-63-2

[46] *Jörg Kloss, Robert Rockwell, Kornel Szabo, Martin Duchrow:* VRML97: Der neue Standard für interaktive 3D-Welten im World Wide Web; Bonn: Addison Wesley, 1998; ISBN 3-8273-1187-X

[47] *James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes:* Computer Graphics: Principles and Practice; Addison-Wesley, 1996

[48] *HTML:* http://www.javasoft.com/products/java-media/3D/.. forDevelopers/j3dguide

[49] *HTML:* http://www.opengl.org/

[50] *HTML:* http://sun.com/desktop/java3d/

[51] *HTML:* http://java.sun.com/products/java-media/3D/

[52] *HTML:* http://www.web3d.org/WorkingGroups/vrml-java3d/

[53] *HTML:* http://www.ee.uwa.edu.au/ braunl/aero/

[54] *HTML:* http://www.aero-simulation.de/