Fenster zum Prozess: ein Operateursarbeitsplatz zur Überwachung und Kontrolle von kooperativem Tracking

Diplomarbeit

im Fachgebiet Informatik

Lehrstuhl: Softwaretechnik

eingereicht am	Institut für Informatik Mathematisch-Naturwissenschaftlichen Fakultät II Humboldt-Universität zu Berlin
von:	Hermann Schwarz
Matrikelnummer:	176887
Erstgutachter:	Prof. Dr. sc. nat. Klaus Bothe
Zweitgutachter:	Prof. Dr. sc. nat. Hartmut Wandke
Betreuer:	Dipl.-Psych. Jens Nachtwei

Hermann Schwarz
Josef-Orlopp-Str. 19
10367 Berlin

hschwarz@informatik.hu-berlin.de

ZUSAMMENFASSUNG

Im Rahmen des Projektes „Arbeitsteilung Entwickler-Operateur“ (ATEO) wird ein Softwaresystem „Socially Augmented Microworld“ (SAM), welches in der Programmiersprache Smalltalk vorliegt, für experimentelle Zwecke benutzt. Diese Software simuliert ein komplexes System, das vom Menschen (Operateur) gesteuert werden soll. Weiterhin werden für Experimente in der Psychologie am Lehrstuhl für Ingenieurpsychologie an der Humboldt Universität zu Berlin (HUB) zum einen eine Softwarekomponente „Operateursarbeitsplatz“ (OA) herangezogen, welche im Rahmen der vorliegenden Arbeit entwickelt wird. Zum anderen wird für experimentelle Zwecke von ATEO innerhalb einer weiteren Diplomarbeit eine automatische Assistenz entwickelt.

Die vorliegende Arbeit beschreibt die Untersuchungen und Schlussfolgerungen aus der Sicht der Software-Ergonomie betreffend des zu entwickelnden Operateursarbeitsplatzes. Aus den durchgeführten Untersuchungen und hierarchischen Aufgabenanalyse des Operateursarbeitsplatzes wird ein Konzept zur Visualisierung der Systemvariablen eines komplexen Systems und Darstellung dieser Visualisierung am Operateursarbeitsplatz entwickelt. Darüber hinaus werden die Steuerelemente entwickelt, welche die Eingriffe in das komplexe System ermöglichen. Das Ziel ist dabei, den Operateur bei seinen Aufgaben effizient zu unterstützen.

Darüber hinaus wird diskutiert, in welcher Phase der Softwareentwicklung die software-ergonomische Analyse stattfinden soll, um die Usability im allgemeinen und die Situationsbewusstsein (SB) des Operateurs im spieziellen Fall im hohen Maße zu unterstützen.

Der praktische Teil der Arbeit umfasst die Entwicklung einer Softwarekomponente für den Operateur, um einerseits eine möglichst breite Sicht auf die Systemparameter des komplexen Systems und andererseits die Eingriffe in das komplexe System zu ermöglichen. Dabei wird versucht, das entwickelte Konzept zur Gestaltung eines Operateursarbeitsplatzes in der Programmiersprache Smalltalk unter der Umgebung Squeak umzusetzen.

Zeitgleich zur vorliegenden Arbeit wird eine automatische Prozessführung des gleichen Systems entwickelt. Die praktischen und theoretischen Ergebnisse der vorliegenden Arbeit werden im Rahmen des ATEO-Projektes zur Beantwortung der Frage beitragen, welche Aufgaben durch die Automatik und welche vom Operateur effizienter und fehlerfreier erledigt werden.

ABSTRACT

Within the project „Working Allocation Developer Operator“ (ATEO) is a software system in experimental purposes. This software is programmed in *Smalltalk* and simulates a complex system wich is controlled by a Operator.

The present work describes the studies and conclusions from the standpoint of software ergonomics regarding the Operator Workstation (Operateursarbeitsplatz, OA). From the investigations carried out and hierarchical task analysis of the Operator workplace is developed a concept for visualization of the system variables of a complex system and display this visualization at Operator Workstation. In addition, the controls are developed by the interference in the complex system. The goal is to support the Operator in carrying out its duties efficiently.

Moreover, discussed at what stage of software development the software ergonomic analysis should take place to ensure usability in general and the situation Awareness of the Operator in special case in the high degree of support.

The practical part of the work involves the development of a software component for the Operator to develop the widest possible view of the system parameters of the complex system and other interventions in the complex system. It is trying to implement the developed concept for the design of the Operator Workstation (Operateursarbeitsplatz, OA) in Smalltalk programming language under the environment Squeak.

At the same time to this work will be developed an automatic process of the same system. The practical and theoretical results of this work will add to the ATEO project to help answer the question, what tasks through the automatic and what by the Operator to be done in more efficient way.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

DEFINITIONEN

FORMELZEICHEN UND ABKÜRZUNGEN

AKE	Automatische kriterienbasierte Evaluierung
ATEO-Projekt	Das Forschungsprojekt „Arbeitsteilung Entwickler-Operateur“
CEM	Concurent Engineering Modell
GUI	Graphical User Interface
HTA	Hierarchical Task Analysis / Hierarchische Aufgabenanalyse
OA	Operateursarbeitsplatz
OOA	Objektorientierte Analyse
OOD	Objektorientiertes Design
prometei	Prospektive Gestaltung von Mensch-Technik-Interaktion
SA	Strukturierte Analyse
SB	Situationsbewusstsein (Situation Awareness)
SAM	Socially Augmented Microworld
SOA	Service Orientierte Architektur
SOAP	Simple Object Access Protocol

1

EINLEITUNG

1.1 MOTIVATION

Operateure komplexer Systeme in der Prozessindustrie, Luftfahrt und Medizin benötigen Technologien, welche eine effiziente Methode zur Überwachung und Kontrolle von im komplexen System verfügbaren Informationen liefern.

Der Grund einer solchen Notwendigkeit ist eine ständig wachsende Kluft zwischen Informationsmenge, welche von zu überwachenden Systemen generiert wird, und der benötigten Informationsmenge (Endsley et al. 2003). Die Informationskluft kann dabei folgende zwei Extremwerte annehmen: Es werden im abnormalen Betriebszustand zu viele Informationen gleichzeitig angezeigt oder es fehlen im Normalbetrieb wegen einer gezielten Informationsverdichtung die Detailinformationen.

Die gesuchte Methode sollte dem Operateur das Verständnis darüber was im zu überwachenden komplexen System abläuft, im hohen Maße anbieten. Dabei soll die Überwachung des Systems in Echtzeit stattfinden. Gleichzeitig soll aber dem Operateur ermöglicht werden, schnell und den Zielen entsprechend in das System einzugreifen.

1.2 ZIEL DER ARBEIT

Das Projekt „Arbeitsteilung Entwickler-Operateur"(ATEO) ist ein Beitrag des Lehrstuhls für Ingenieurpsychologie und Kognitive Ergonomie der Humboldt-Universität zu Berlin zum Graduiertenkolleg „prometei"(Prospektive Gestaltung von Mensch-Technik-Interaktion), das organisatorisch am Zentrum Mensch-Maschine-Systeme der Technischen Universität Berlin angesiedelt ist.

Der Schwerpunkt des Projektes liegt gegenwärtig darauf, eine asynchrone Arbeitsteilung bei Entwicklung und Benutzung komplexer, dynamischer, technischer Systeme zwischen folgenden Gruppen zu untersuchen:

1. Menschen, die Systeme, insbesondere Assistenzsysteme, planen und implementieren (Entwickler)
2. Menschen, die diese Systeme benutzen (Operateure)

Der Entwickler besitzt normalerweise Kenntnis über das zu implementierende oder bereits bestehende System. Allerdings ist die Situation, in der dieses System benutzt wird, dem Entwickler

im Allgemeinen weniger bekannt. Insbesondere fehlt dem Entwickler oft die Kenntnis über die abnormen und daher schwer antizipierbaren Situationen. Die Überwachung und die Steuerung des Systems erfolgt aus der Perspektive des Entwicklers indirekt durch die implementierte Automatik.

Der Operateur besitzt dagegen nicht nur eine umfangreichere Wissensbasis über die Situation, sondern auch einen Einblick in das Systemverhalten zur Laufzeit des Systems. Der Operateur kann somit besser erkennen, ob das System das gewünschte Verhalten besitzt.

Ziel des Projektes ist zu untersuchen, welche Informationen einerseits dem Entwickler der automatischen Steuerung gegeben werden und andererseits welche Informationen dem Operateur angezeigt werden müssen, um damit die Erkennbarkeit und kurzfristige Antizipation von Systemkonflikten zu verbessern. Als Systemkonflikte gelten dabei Störfälle oder abnorme Situationen.

Eine Verbesserung der Prozessführungsleistung zählt ebenfalls zu den Zielen des ATEO-Projektes.

Für die experimentelle Untersuchung des oben beschriebenen Sachverhaltes wird die so genannte Socially Augmented Microworld (SAM) in Form einer Software herangezogen. Bei SAM handelt es sich um eine Versuchsumgebung, in welcher eine Trackingaufgabe kooperativ von zwei Personen, die als Mikroweltbewohner (MWB) bezeichnet werden, durchgeführt wird. Die Trackingaufgabe besteht darin, ein Objekt auf einer Strecke möglichst genau und so schnell wie möglich entlang zu führen.

MWB stellen eine stochastische Komponente von SAM dar und dienen damit zur Erhöhung der Komplexität des Systems. Diese zwei Personen sind demzufolge nur ein Teil des zu kontrollierenden Systems. SAM fungiert als ein komplexes System, welches vom Operateur überwacht und kontrolliert wird. In weiteren Schritten des ATEO-Projektes wird SAM von Entwicklern mit einer Automatik versehen. Das Ziel dieser Projektphase ist, die Leistung der automatischen Steuerung mit der von Operateuren zu vergleichen. Das Ergebnis dieses Vergleiches wird anschließend die Frage beantworten, welche Funktionen durch den Operateur und welche durch die automatische Überwachung und Steuerung übernommen werden können. Die Einordnung der vorliegenden Arbeit in das ATEO-Projekt wird in der Abbildung (Arbeitspaket O2) grafisch dargestellt. Der Vergleich der automatischen Steuerung mit der Steuerung durch einen Operateur wird im Rahmen des abschließenden Arbeitspaketes A4 durchgeführt.

Bisher erwies es sich für den Operateur als schwierig, seine Aufgabe zu erledigen. Der Grund ist eine fehlende oder nicht ausreichende Möglichkeit der Überwachung bestimmter Systemparameter wie Geschwindigkeit und Qualität der Prozesse innerhalb des zu betrachtenden Systems oder das Maß der Beanspruchung

von MWB. Im Rahmen des Experimentes werden die Prozessparameter an den MWB und ihrem Verhalten gemessen. Bisher wurden diese Parameter lediglich in eine Logdatei protokolliert. Da der zentrale Aspekt für den Operateur das „Fenster zum Prozess“ ist, wird im Rahmen eines SAM-Experimentes das Ziel verfolgt, die Prozessparameter zur Laufzeit beobachten und analysieren zu können.

Die SAM-Software wurde unter der Smalltalk-Implementierung und Entwicklungsumgebung Squeak entwickelt. Im Rahmen der Diplomarbeit werden Möglichkeiten des Squeak-Systems im Hinblick auf Überwachung von Echtzeitdaten und Steuerung vom SAM durch den Operateur untersucht. Für den Entwurf und die Implementierung werden software-ergonomische Metriken und Grundsätze herangezogen. Weiterhin wird eine hierarchische Aufgabenanalyse (Hierarchical Task Analysis, HTA) durchgeführt, da sie eine der Voraussetzungen für eine aufgabenangemessene Benutzungsschnittstelle ist (Herczeg 1994). Darüber hinaus wird eine objektorientierte Analyse (OOA) und darauf aufbauende objektorientierte Modellierung des Operateursarbeitsplatzes durchgeführt, welche die Basis für den darauf folgenden Entwurf und die Implementierung darstellen. Als Ergebnis des praktischen Anteils der Arbeit wird die vorliegende Software SAM um eine Komponente für den Operateursarbeitsplatz (OA) erweitert. Beide Softwarekomponenten werden eine verteilte Anwendung darstellen und im Rahmen einer weiteren Diplomarbeit von Nicolas Niestroj vernetzt.

1.3 GLIEDERUNG DER ARBEIT

Zunächst werden im Kapitel 2 Definitionen, Modelle und Konzepte eingeführt, die für die Untersuchungen und Schlussfolgerungen der Arbeit relevant sind.

Im zentralen Kapitel der vorliegenden Arbeit (Kap. 3) werden nach einer detaillierten Definition des Begriffes *Software-Ergonomie* die software-ergonomischen Grundsätze und Metriken vorgestellt. Weiterhin werden im Kapitel 3 Ansätze zur Bewertung von grafischen Benutzungsschnittstellen (engl. Graphical User Interface, Abk. GUI) vorgestellt und bewertet. Die GUI-Elemente des zu entwickelnden Systems werden formal definiert. Damit wird versucht, eine Grundlage für interdisziplinäres Arbeiten an der Entwicklung von GUI zu schaffen. Zusätzlich kann eine solche formale Beschreibung für eine GUI-Evaluierungsmethode herangezogen werden. Abschließend werden die Vorgehensmodelle hinsichtlich des Einflusses auf die qualitativen software-ergonomischen Eigenschaften untersucht.

Im Kapitel 4 wird zunächst die durchgeführte Systemanalyse vorgestellt. Dabei werden sowohl die Anwendungsfälle definiert als auch die Anforderungen an den Operateursarbeitsplatz (OA)

präzisiert. Durch die Anwendungsfälle werden Aktionen spezifiziert, welche vom Subjekt OA zum Erreichen von Zielen des Operateurs ausgeführt werden müssen. Nachdem die durchgeführte HTA und das erstellte OOD-Modell vorgestellt werden, werden die relevanten Aspekte der Implementierung wie Überwachung oder Kontrolle des Fahrobjektes beschrieben.

Kapitel 5 dient zum Zusammenfassen der im Rahmen der vorliegenden Arbeit gewonnenen Ergebnisse. Weiterhin werden in diesem Kapitel Möglichkeiten zur Weiterentwicklung und alternative Implementierungsmöglichkeiten diskutiert.

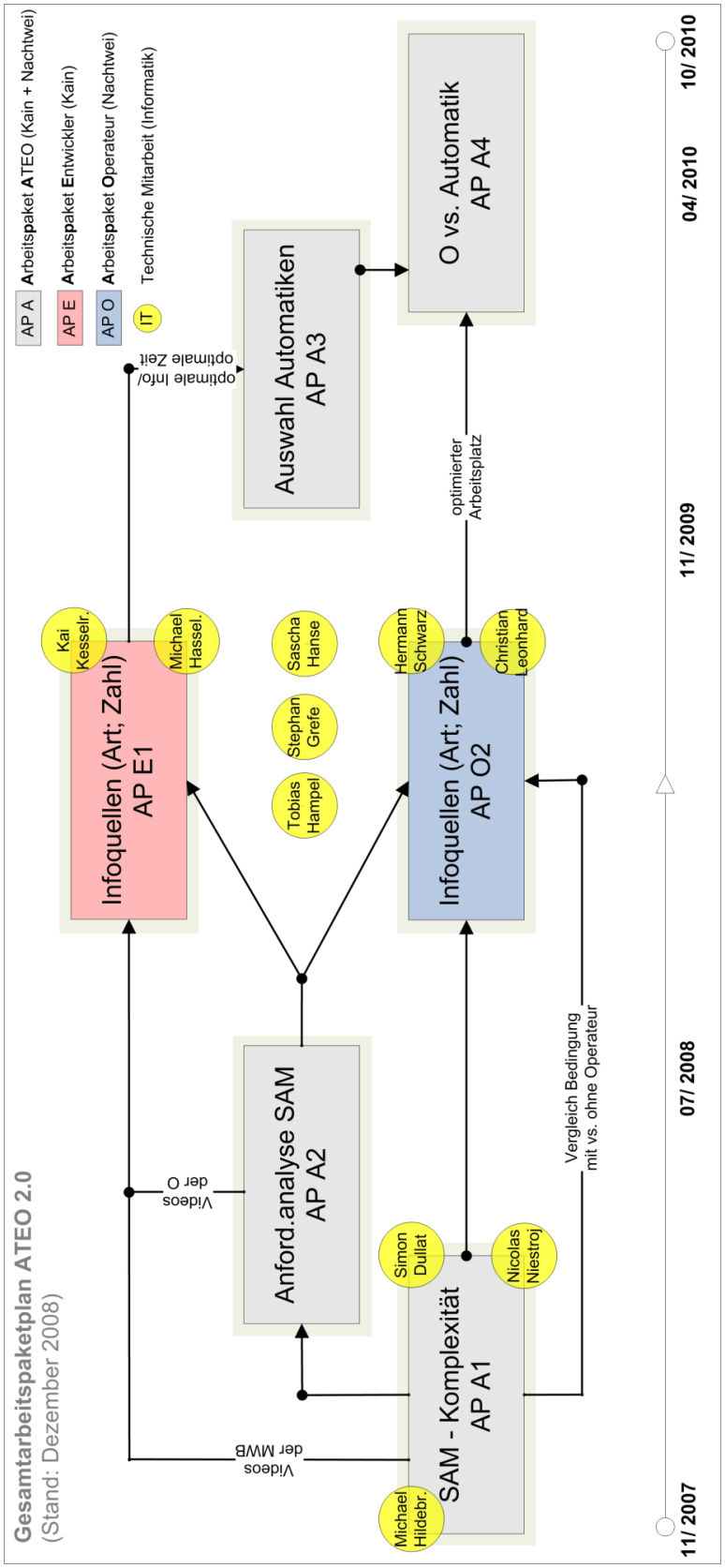

Abbildung 1: Einordnung der Arbeit am Operateursarbeitsplatz im ATEO-Projekt

2

GRUNDLAGEN

In diesem Kapitel werden die grundlegenden Begriffe definiert und die für diese Arbeit relevanten Konzepte, Technologien und Begriffe erläutert. Damit wird die Grundlage zum Verständnis des in der Arbeit zu erarbeitenden Themas geschaffen.

Nach einer kurzen Einführung in Squeak wird das *SOA*-Konzept (Serviceorientierte Architektur) vorgestellt, da die Vernetzung von Socially Augmented Microworld (SAM) und Operateursarbeitsplatz (OA) nach diesem Konzept implementiert wird.

Weiterhin wird der Begriff des *benutzerorientierten Designs* definiert. Die Notwendigkeit des benutzerorientierten Designs in der Software-Entwicklung von Mensch-Maschine-Systemen wird begründet. Als wichtiges Instrument für eine benutzerorientierte Softwareentwicklung, welches aus dem Bereich der Ingenieurpsychologie stammt, wird *Hierarchische Aufgabenanalyse* (Hierarchical Task Analysis, HTA) vorgestellt. HTA wird anschließlich mit einem ähnlichen Konzept aus der Informatik *Strukturierte Analyse* (SA) verglichen.

Dieses Kapitel wird mit der Definition eines Begriffes abgeschlossen, welcher als ein Indikator für die Güte der Software-Ergonomie dienen kann. Es handelt sich hierbei um den Begriff *Situationsbewusstsein* (SB).

2.1 SQUEAK

Das Projekt Squeak wurde 1995 von Apple ins Leben gerufen. Squeak stellt eine freie Implementierung der objekt-orientierten Sprache Smalltalk dar. Die Entwicklungsumgebung von Squeak bietet eine Vielzahl an Entwicklungswerkzeugen wie Debugger, Method-Finder und verschiedene Browser. Die Squeak-Entwicklungsumgebung ist in Smalltalk geschrieben und kann daher bequem geändert und erweitert werden.

2.1.1 *Morphic*

Morphic ist das direkt manipulierbare User Interface (UI) von Squeak. Alle Bestandteile des Morphic-UI werden als Morphe bezeichnet. Der Begriff *Morph* kommt aus dem Griechischen und heißt Form. Morphe sind Objekte von Squeak, die eine visuelle Darstellung besitzen und in Form und Position geändert werden können. Alle Objekte, die auf einem Squeak-Desktop sichtbar sind, sind Morphe oder eine Zusammensetzung solcher.

In der Abbildung 2 ist beispielsweise ein *SystemWindow* dargestellt. Die Klasse *SystemWindow* ist eine Unterklasse von *Morph* (Abb. 3). Mit Hilfe der Klasse *SystemWindow* können editierbare Texte dargestellt werden. Ein weiteres Beispiel einer Unterklasse von *Morph* ist in der Abbildung 4 dargestellt. Hierbei handelt es sich um die integrierte Entwicklungsumgebung (IDE) von Squeak.

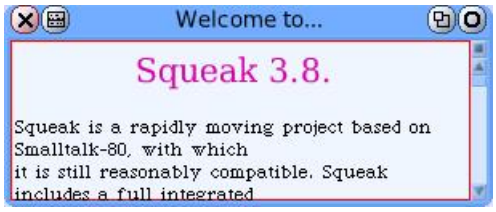

Abbildung 2: *SystemWindow*: eine Unterklasse von *Morph*

Abbildung 3: *SystemWindow*-Klassenhierarchie

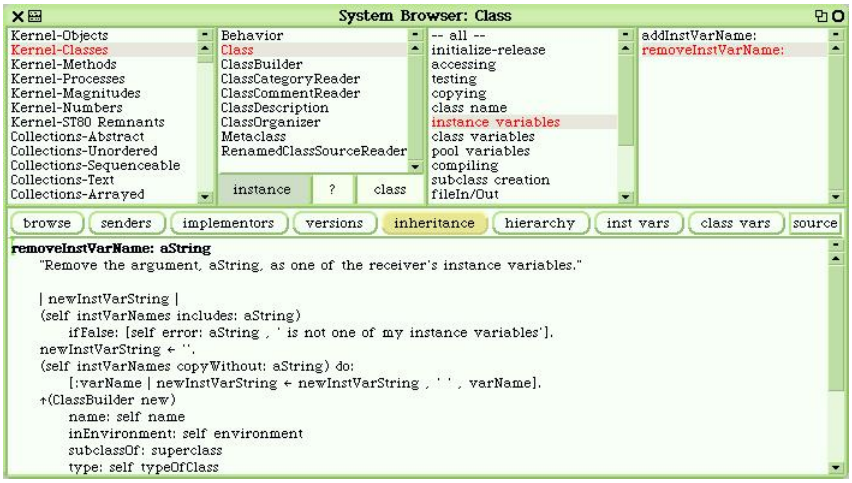

Abbildung 4: *SystemWindow*: eine Unterklasse von *Morph*

2.1.2 *Squeak API*

Die Programmiersprache Smalltalk ist eine objekt-orientierte Sprache, so dass die Eigenschaften der bereits implementierten

Klassen durch neue Klassen geerbt werden können. Zur Erleichterung der Entwicklung eigener Software existiert eine Dokumentation der Programmierschnittstelle (API) von Squeak[1]. Leider ist in dieser Dokumentation nicht jede Methode beschrieben.

2.2 SOA

Der im Rahmen der vorliegenden Arbeit zu entwickelnde Operateursarbeitsplatz (OA) wird mit der Socially Augmented Microworld (SAM) vernetzt. Die Vernetzung wird mit Hilfe der serviceorientierter Architektur (SOA, Def. 2.2.1) umgesetzt.

Definition 2.2.1 (SOA) *Eine Menge von Softwarekomponenten, welche über bereitgestellte Schnittstellen aufgerufen werden können. Die Beschreibung der Schnittstellen kann veröffentlicht werden, um die Softwarekomponenten auffindbar zu machen (Booth und Haas 2004).*

Technisch wird die Vernetzung von OA und SAM mit Hilfe von SOAP[2] (ursprünglich *Simple Object Access Protocol*) umgesetzt. Squeak bietet einige Implementierungen von SOAP. Die Einzelheiten zur Vernetzung von OA und SAM werden in der Diplomarbeit von Nicolas Niestroj dargestellt.

2.3 BENUTZERORIENTIERTES SOFTWAREDESIGN

Traditionell werden Softwaresysteme aus der technologieorientierten Perspektive entwickelt. Es werden dabei die Anforderungen an die Software erfüllt oder teilweise erfüllt, indem die in den Softwareanforderungen beschriebenen Funktionen implementiert werden. So entwickelte Softwaresysteme, welche darüber hinaus für eine Interaktion mit Menschen konzipiert werden, können im Allgemeinen nicht fehlerfrei bedient werden und arbeiten nicht optimal. Der Grund dafür ist der mangelnde Bezug zum Benutzer einer Software. Beispielsweise werden Begrenzungen in der Informationsverarbeitung eines Benutzers nur gering beachtet (Endsley et al. 2003).

Mit Hilfe eines benutzerorientierten Designs wird dagegen primär versucht, die Fehleranfälligkeit des Mensch-Maschine-Systems zu reduzieren, indem eine Nutzungsschnittstelle (engl. User Interface, Abk. UI) unter Berücksichtigung der Bedürfnisse und Fähigkeiten des Benutzers entwickelt wird. Darüber hinaus wird dabei versucht, das Akzeptanz- und Zufriedenheitsniveau, die Sicherheit, und die Produktivität des Mensch-Maschine-Systems zu erhöhen (Endsley et al. 2003).

1 Squeak API Dokumentation, www.oldenbuettel.de/squeak-doku/class_index.html
2 SOAP, www.w3.org/TR/soap

Ein zentrales Werkzeug zur Realisierung eines benutzerorientierten Softwaredesigns ist die Hierarchische Aufgabenanalyse (Hierarchical Task Analysis, HTA).

2.4 HIERARCHISCHE AUFGABENANALYSE (HTA)

Im Rahmen der vorliegenden Arbeit wurde eine hierarchische Aufgabenanalyse (*Hierarchical Task Analysis, HTA*) des Operateursarbeitsplatzes durchgeführt. Bevor die Zweckmäßigkeit einer solchen Analyse erläutert wird, folgt eine Definition von HTA.

Definition 2.4.1 (HTA) *Ein abstraktes Modell eines komplexen Systems, welches darauf basiert, die von Menschen durchgeführten Aufgaben (Tasks, Goals) innerhalb dieses Systems in Unteraufgaben (Subgoals) zu untergliedern.*

Anhand eines solchen Modells können anschließend die Möglichkeiten einer Computer-Unterstützung bei der Durchführung dieser Aufgaben untersucht werden. (Barfield 2004).

Ein Beispiel einer HTA wird in der Abbildung 5 dargestellt. Alle Aspekte von HTA werden im Weiteren anhand dieser Abbildung erläutert.

Die in solchen Modellen beschriebenen Unteraufgaben (Subgoals) haben oft einen prozeduralen Charakter, das heißt, die Unteraufgaben werden sequenziell ausgeführt. Allerdings ist dies nicht immer der Fall. Daher wird für jede Aufgabe, die in weitere Unteraufgaben untergliedert wird, ein Plan beschrieben (Abb. 5, a). In den Plänen werden Kontrollkonstrukte zum Ausführen der Unteraufgaben definiert (Stanton 2005).

In einer HTA werden Aufgaben oder Ziele des Nutzers hierarchisch beschrieben, indem sie beginnend mit dem Hauptziel des Nutzers immer weiter verfeinert werden. Begonnen wird mit dem Hauptziel des Nutzers. Dabei befindet sich dieses Ziel auf der höchsten Abstraktions- und Hierarchieebene (Abb. 5, b). Die Aufgaben werden immer weiter verfeinert. Das Abstraktionsniveau zu bestimmen, an dem die Verfeinerung der Aufgaben gestoppt werden muss, ist eine Herausforderung bei der Erstellung der HTA. Es kann dabei folgende Heuristik benutzt werden (Stanton 2005) :

$$
\begin{aligned}
&\mathbf{P} - \text{Wahrscheinlichkeit für Auftreten eines Fehlers} \\
&\text{beim Ausführen einer Aufgabe} \\
&\mathbf{C} - \text{Kosten des Fehlers} \\
&\mathcal{F} = \mathbf{P}x\mathbf{C}
\end{aligned}
\tag{2.1}
$$

Algorithmus:
WHILE $\mathcal{F}$ nicht akzeptabel
DO Verfeinerung der Aufgaben

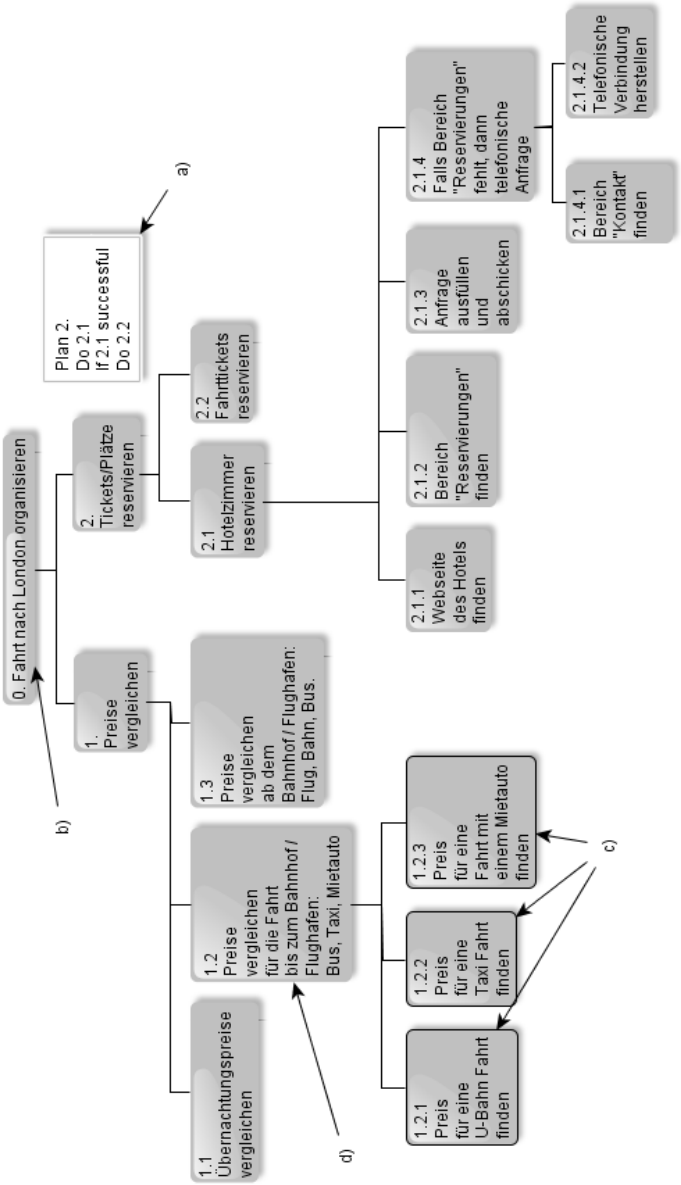

Abbildung 5: Beispiel einer HTA

In den meisten Fällen sind P und C aus der Formel 2.1 schwierig zu berechnen, so dass eine Approximation benutzt werden muss.

Als ein weiteres Abbruchkriterium kann folgende Bedingung verwendet werden: sobald die Aufgabe für alle beteiligten Entwickler und Systemexperten klar definiert ist und programmiertechnisch umgesetzt werden kann, kann die HTA abgebrochen werden (Piso 1981).

Primäres Ziel von HTA im Rahmen eines User Interface Designs ist ein hinsichtlich der Usability besseres Design zu ermöglichen. Erstens kann anhand HTA sichergestellt werden, dass das entwickelte Design alle in der HTA definierte Aufgaben unterstützt. Weiterhin können alternative Designs verglichen werden. Dabei wird das Design, welches die Aufgaben in den Blättern des HTA-Baumes (Abb. 5, c) implementiert, mit dem Design verglichen, welches die Implementierung der Aufgaben in den Elternknoten von Blättern (Abb. 5, d) enthält. Im Allgemeinen ist das Design, welches die Aufgaben in den Knoten einer niedrigeren Tiefe (höher im HTA-Baum) implementiert, benutzerfreundlicher. Die Begründung dafür ist die Abnahme der Komplexität der Mensch-Maschine Interaktion, wenn die Knotentiefe der zu implementierenden Aufgabe abnimmt. Denn jeder HTA-Knoten besitzt mindestens einen Kinderknoten. In der Praxis sind es aber mindestens zwei Kinderknoten, was auf die Eigenschaft von der HTA zurückzuführen ist, Aufgaben immer weiter zu zerlegen. So ist das Maß und die Komplexität der Mensch-Maschine Interaktion eines Knoten (Elternknoten) im Allgemeinen niedriger als die Kumulation beider Größen über mehrere Knoten (Kinderknoten). Allerdings sind die Aufgaben, die sich in der niedrigeren Tiefe des HTA-Baumes befinden, schwieriger zu implementieren.

Im Idealfall würde der Benutzer nur eine Aktion mit Hilfe des entwickelten Systems ausführen, um das Hauptziel zu erreichen, wenn es möglich wäre ein solches System zu entwickeln. In diesem Fall würde der Entwickler die Wurzel von der HTA für das Systemdesign verwenden (Abb. 5, b).

Ein weiterer Vorteil neben der Möglichkeit der Verbesserung der Software-Ergonomie von der HTA ist die Möglichkeit einer interdisziplinären Arbeit am zu analysierenden und zu entwickelnden System, denn an der Entwicklung eines Softwaresystems arbeiten meistens nicht nur Informatiker, so dass im Idealfall solche Entwurfs- und Modellierungskonzepte benötigt werden, die von allen Beteiligten verstanden werden.

Ein ähnliches in der Informatik bekanntes Modellierungskonzept, welches eine hierarchische Sicht auf die durchzuführenden Aufgaben, Aktivitäten oder Prozesse bietet, ist die *Strukturierte Analyse* (SA). Darüber hinaus werden innerhalb der SA Datenflüsse beschrieben, welche von beschriebenen Aktivitäten transformiert werden. Damit wird die Interaktion zwischen Aktivitäten

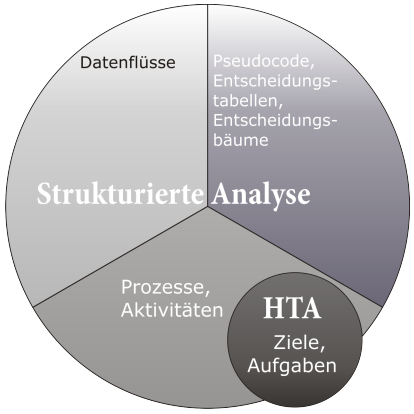

Abbildung 6: Aspekte von SA und HTA

beschrieben. Die Datenflüsse können ebenfalls wie Aktivitäten hierarchisch verfeinert werden. Die Blätter des Baumes, welcher durch die Verfeinerung der Aktivitäten entsteht, sind die elementaren Prozesse. Sie werden in den *Minispezifikationen* mit Hilfe des Pseudocodes, Entscheidungstabellen oder -bäumen beschrieben (Balzert 2000).

In der Abbildung 6 wird grafisch gezeigt, dass SA und HTA eine nichtleere Schnittmenge bilden. HTA ist dabei aus zweierlei Gründen keine Teilmenge von SA. Einerseits wird versucht die mit Hilfe von HTA beschriebenen Aktivitäten nicht als Prozesse sondern als Aufgaben oder Ziele zu definieren. Das heißt, dass HTA primär die Ziele beschreibt, die ein Nutzer erreichen möchte, und nicht unbedingt die Prozesse, die zum Erreichen dieses Ziels führen. Zweitens beginnt die Verfeinerung der Aufgaben/Aktivitäten unter HTA auf einem höheren Abstraktionsniveau als bei der SA. Der Grund dafür ist wiederum die zielorientierte Natur von der HTA. Ziele befinden sich im Allgemeinen auf einem höheren Abstraktionsniveau als die Prozesse, die zum Erreichen der Ziele führen.

Aufgrund der Komplexität und Mannigfaltigkeit der verwendeten Konzepte von der SA ist diese Modellierungsmethode für eine software-ergonomische Analyse in einem interdisziplinären Team weniger passend als die HTA. Die im Rahmen der vorliegenden Arbeit durchgeführte HTA wird im Abschnitt 4.2 vorgestellt.

2.5 SITUATIONSBEWUSSTSEIN

Während HTA als ein Hilfsmittel zur Verbesserung der Software-Ergonomie dienen kann, ist der Begriff *Situationsbewusstsein* einer der Indikatoren für die Güte der Software-Ergonomie.

Der Begriff *Situationsbewusstsein* (SB) (engl. *Situation Awareness*, SA) kommt aus dem Gebiet Human Factors und entstand in

den 1980ern, als Ingenieure der Luft- und Raumfahrtindustrie versuchten zu verstehen, wie Piloten große Informationsmengen erfassen, sortieren und abarbeiten. Es wurde auch nach Methoden gesucht, welche diese mentalen Prozesse der Piloten optimieren.

Allgemein beschreibt der Begriff *Situationsbewusstsein* den Prozess der Aufnahme und Verarbeitung von Informationen aus der Umwelt durch Menschen, beispielsweise durch einen Operateur. Es gibt viele Definitionen von SB, wobei einige an bestimmte Domänen gebunden sind und die anderen genereller Natur sind. Eine generelle Definition von SB (Def. 2.5.1), die in vielen Domänen anwendbar ist stammt von Endsley (Endsley 1988).

Definition 2.5.1 (SB) ***Wahrnehmen*** *von Elementen einer bestimmten räumlichen Umgebung innerhalb einer bestimmten Zeit, das* ***Verständnis*** *der Bedeutung dieser Elemente sowie* ***Prognose*** *des Zustandes dieser Elemente in der nächsten Zukunft (Endsley 1988).*

Wahrnehmen*,* ***Verstehen*** *und* ***Prognose*** *werden dabei als aufeinander aufbauende Ebenen betrachtet. Die Ebene des* ***Wahrnehmen*** *ist grundlegend. Die meisten Fehler in SB sind auf ein fehlerhaftes Wahrnehmen zurückzuführen. Auf der nächsthöheren Ebene, der Ebene des* ***Verstehens*** *werden die wahrgenommenen Informationen nach ihrer Relevanz bezüglich zu erreichender Ziele bewertet. Auf dem höchsten Niveau befindet sich die Ebene der* ***Prognose****. An dieser Stelle werden künftige Ereignisse und die Dynamik der Situation antizipiert und somit die Voraussetzung für Handlungsentscheidungen geschaffen.*

Das Maß der SB zeigt somit, wie präzise das reale Modell, welches mit Hilfe einer Software überwacht und gesteuert wird, und das mentale Modell des realen Models übereinstimmen (Banbury und Tremblay 2004). Die Gleichung 2.2 drückt diesen Sachverhalt aus.

$$\begin{aligned} &\mathbf{R} - \text{Reales Modell des durch eine Software} \\ &\text{zu überwachenden und zu kontrollierenden Systems} \\ &\mathbf{M}(\mathbf{R}) - \text{Mentales Modell des realen Models} \\ &\mathcal{M}(\text{SB}) - \text{Maß von SB} \\ &\mathcal{M}(\text{SB}) \equiv |\mathbf{R} \cap \mathbf{M}(\mathbf{R})| \end{aligned} \tag{2.2}$$

In der Gleichung 2.2 ist es ersichtlich, dass das Maximum von $\mathcal{M}(\text{SB})$ erreicht wird, wenn die Mengen $\mathbf{R}$ und $\mathbf{M}(\mathbf{R})$ gleich sind. $\mathcal{M}(\text{SB})$ erreicht sein Minimum, wenn die Schnittmenge $\mathbf{R} \cap \mathbf{M}(\mathbf{R})$ leer ist.

Es existiert eine Vielzahl an Methoden, um SB zu messen. Anhand der Ergebnisse dieser Messungen können Aussagen über die Qualität eines GUI getroffen werden. Im Kapitel 3 werden die Messmethoden hinsichtlich der Aufnahme in ein Softwareentwicklungsprozess untersucht.

Die Qualität des GUI kann ebenfalls anhand der mentalen Beanspruchung des Nutzers gemessen werden. Daher werden im Kapitel 3 auch die Möglichkeiten zum Messen der mentalen Beanspruchung und der Komplexität von GUI untersucht.

3 THEORETISCHE AUSEINANDERSETZUNG

Das Ziel dieses Kapitels ist eine Untersuchung von nutzerorientierten Methoden zur Entwicklung einer grafischen Benutzungsschnittstelle (engl. Graphical User Interface, Abk. GUI) eines vom Operateur zu kontrollierenden Systems.

Zur Erreichung dieses Ziels wird zunächst im Abschnitt 3.1 Definition des Begriffes *Software-Ergonomie* vorgestellt. Es wird ein Modell gezeigt, welches alle Software-Qualitätsmerkmale zusammenfasst. Einige Software-Qualitätsmerkmale können als Bewertungskriterien oder Elemente eines Bewertungskriteriums für Software-Ergonomie angesehen werden. Das zentrale Bewertungskriterium der Software-Ergonomie, die Gebrauchstauglichkeit wird definiert. Darüber hinaus werden Maße dieses Bewertungskriteriums vorgestellt.

Der darauf folgende Abschnitt 3.2 enthält Vergleich und Ausarbeitung software-ergonomischer Richtlinien (Styleguides) und Ergebnisse software-ergonomischer Untersuchungen. Der Überwiegende Teil dieser Ergebnisse stammt aus dem Bereich Ingenieurpsychologie und stellt somit eine der Schnittstellen der vorliegenden Arbeit zu Disziplinen der Psychologie.

Ein relevanter Bestandteil der software-ergonomischen Softwareentwicklung ist die Evaluation des entwickelten GUI. Die Evaluation kann sowohl während der Entwicklung formativ als auch summativ, sobald ein lauffähiger Prototyp vorhanden ist, durchgeführt werden. Beide Ansätze wurden in Ihrer Effektivität bewertet. Die Vorstellung der Ansätze und die Ergebnisse des Vergleiches sind im Abschnitt 3.3 dargestellt.

Ausgehend von den Untersuchungen in den Abschnitten 3.1 und 3.2, technischen Möglichkeiten von Squeak und zur Verfügung stehenden Klassenbibliotheken wird ein Konzept zur objektorientierten Beschreibung eines software-ergonomischen GUI-Entwurfes erarbeitet. Dieses Konzept wird im Unterabschnitt 3.4.1 dargestellt.

Abgeschlossen wird das zentrale Kapitel der vorliegenden Arbeit mit der Vorstellung eines Vorgehensmodells, welches einen Software-Entwicklungsprozess unter Berücksichtigung von software-ergonomischen Kriterien darstellt. Das Vorgehensmodell soll dabei die Erfüllung software-ergonomischer Kriterien im hohen Maße unterstützen.

3.1 SOFTWARE-ERGONOMISCHE ANFORDERUNGEN

Eine der Fragestellungen der vorliegenden Arbeit ist, mit welchen Mitteln ergonomische Software-Systeme systematisch entwickelt werden können. Nach wie vor stellt die Entwicklung solcher Systeme eine Herausforderung dar. Dafür gibt es unterschiedliche Gründe. Der Teildisziplin Software-Ergonomie wird sowohl in der Lehre als auch in der Praxis weniger Aufmerksamkeit geschenkt als den restlichen Disziplinen des Software-Engineering. Formale Methoden zu Evaluierung der ergonomischen Eigenschaften von Software haben oft nur einen konzeptionellen Charakter.

Für die vorliegende Arbeit ist es aber von großer Bedeutung, eine Methode zur Entwicklung ergonomischer Software zu entwickeln. Diese Notwendigkeit ergibt sich aus dem Ziel, dem Operateur eine effektive und effiziente Möglichkeit zum Überwachen und zum Eingreifen in ein komplexes System anzubieten.

3.1.1 *Definition der Software-Ergonomie*

Zunächst soll mit Hilfe der Definition 3.1.2 nach (Balzert 2000) geklärt werden, was unter Software-Ergonomie verstanden wird.

Da *Gebrauchstauglichkeit* als Ziel und Produkt der Software-Ergonomie angesehen wird, beschreibt die Definition 3.1.1 auch den Begriff *Gebrauchstauglichkeit* (Balzert 2000).

Definition 3.1.1 (Gebrauchstauglichkeit) *Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele **effektiv**, **effizient** und **zufriedenstellend** zu erreichen.*

- ***Effektivität** ist die Genauigkeit und Vollständigkeit, mit welcher die Benutzer ein bestimmtes Ziel erreichen.*
- ***Effizienz** ist der eingesetzte Aufwand im Verhältnis zur Effektivität.*
- ***Zufriedenstellung** ist*
  - *die Freiheit von Beeinträchtigungen und*
  - *positive Einstellungen gegenüber der Nutzung des Produkts.*

Definition 3.1.2 (Software-Ergonomie) *Ein Teilbereich von Software-Engineering, der sich mit der Analyse solcher Problembereiche wie*

- *Bedienungsprobleme*
- *Funktionalitätsprobleme*
- *Unterforderungs-, Überforderungs-, Monotonie und Arbeitsintensitätsprobleme*

und mit der menschengerechten Gestaltung von Software-Systemen befasst. Software-Ergonomie verfolgt das Ziel, ***gebrauchstaugliche*** *Software zu entwickeln, welche die Benutzer zur Erreichung ihrer Arbeitsergebnisse befähigt und dabei ihre Belange im jeweiligen Nutzungskontext beachtet.*

Laut (Balzert 2000) ist die Gebrauchstauglichkeit eines Software-Produktes das wichtigste Bewertungskriterium aus der Sicht des Benutzers. Der Grad von Gebrauchstauglichkeit hängt vom jeweiligen Nutzungskontext ab und kann für verschiedene Kontexte und Ziele auch unterschiedliche Werte liefern (Balzert 2000). In unserem konkreten Fall ist der Nutzungskontext bekannt. Die Software soll von einem Operateur zum Überwachen eines Systems und zum Eingreifen in dieses System genutzt werden.

Die Maße der Gebrauchstauglichkeit sind nach Definition 3.1.1 die Software-Qualitätsmermake *Effektivität*, *Effizienz* und *Zufriedenstellung*. Es existieren Modelle, welche alle Software-Qualitätsmerkmale zusammenfassen. Diese Modelle werden gelegentlich auch als FCM-Modelle (factor, criterion, metrics) bezeichnet.

Eines solcher Modelle, welches in (iso 1991) standardisiert ist, wird in der Abbildung 7 dargestellt. Das in dieser Abbildung gezeigte Modell enthält alle Software-Qualitätsmermale, die die Softwarequalität beeinflüssen.

In der Abbildung 7 ist ebenfalls die Beziehung der Software-Qualitätsmermale zur Software-Ergonomie gezeigt, indem nur die Software-Qualitätsmermale repräsentierenden Knoten mit dem Knoten *Ziel der Software-Ergonomie: Gebrauchstauglichkeit* verbunden sind, welche direkt oder indirekt die Gebrauchstauglichkeit beeinflussen. Mit der abgebildeten Beziehung wird gezeigt, welche Teilmenge aller Software-Qualitätsmerkmale zu den Zielen der Software-Ergonomie gehört. Die Merkmale dieser Teilmenge werden als *Qualitätsmerkmale aus Benutzersicht* und die Merkmale der komplementären Teilmenge als *Qualitätsmerkmale aus Entwicklersicht* bezeichnet (Kahlbrandt).

Da in der vorliegenden Arbeit die Aspekte im Vordergrund stehen, die für den Anwender relevant sind, werden im Weiteren nur die *Qualitätsmerkmale aus Benutzersicht* in Hinsicht auf das Erreichen dieser Merkmale untersucht.

3.1.2 *Spezifizierung und Evaluation der Gebrauchstauglichkeit*

Für eine software-ergonomische Softwareentwickling ist es vorteilhaft, die Gebrauchstauglichkeit zu spezifizieren, um sie im Zuge der Testabläufe zu evaluieren. Eine Vorgehensweise, wie bei der Entwicklung die Gebrauchstauglichkeit spezifiziert und evaluiert werden kann, wird in (ENi 1998) beschrieben. Diese Quelle gibt an, welche Anforderungen im Pflichtenheft festgelegt werden müssen, um die Gebrauchstauglichkeit zu spezifizieren:

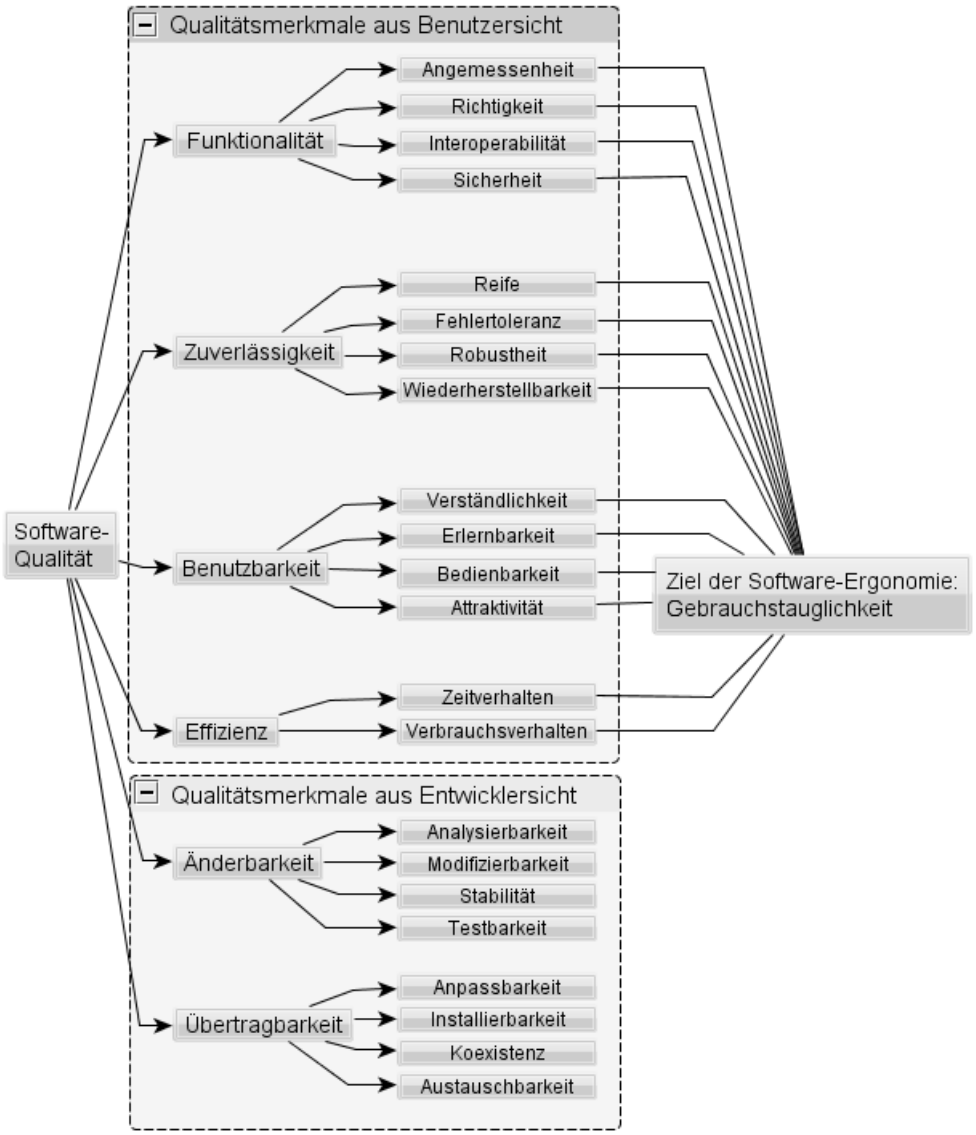

Abbildung 7: Software-Qualitätsmerkmale (iso 1991) mit Beziehung zum Ziel der Software-Ergonomie, der Gebrauchstauglichkeit

1. Beschreibung der Ziele der Softwarenutzung
2. Beschreibung der Arbeitsaufgaben
3. Beschreibung der Nutzermerkmale
4. Beschreibung der Arbeitsmittelmerkmale und der Merkmale der physischen Umgebung
5. Festlegung der Maße der Gebrauchstauglichkeit

Im folgenden wird beschrieben, wie die oben genannten Punkte spezifiziert werden können.

Ziele der Softwarenutzung und Arbeitsaufgaben

Das Ziel der Produktnutzung wird im Abschnitt „Zielbestimmung“ des Pflichtenheftes definiert. Die Teilziele werden mit Hilfe von Geschäftsprozessen spezifiziert (Balzert 2000). Auch die Arbeitsaufgaben sollen mit Hilfe von Geschäftsprozessen zielbezogen beschrieben werden. Darüber hinaus können Geschäftsprozesse durch eine Beschreibung der Aufgabenmerkmale ergänzt werden, welche die Gebrauchstauglichkeit beeinflüssen können (Balzert 2000). Es kann beispielsweise die Dauer oder die Häufigkeit der Durchführung einer Aufgabe angegeben werden.

Eine HTA kann als ein ergänzender oder sogar alternativer Ansatz zum Spezifizieren der Ziele der Softwarenutzung und der Arbeitsaufgaben angesehen werden. Geschäftsprozesse und die HTA wurden in einigen Aspekten verglichen, um die Entscheidung für das eine oder das andere Konzept zu erleichtern. In der Tabelle 1 ist der Vergleich der Aspekte der HTA und Geschäftsprozesse zusammengefasst.

Tabelle 1: Vergleich von HTA und Geschäftsprozessen

Aspekt	HTA	Geschäftsprozesse / Geschäftsprozessdiagramm
Grafische Darstellung von Zielen/Aufgaben	Ziele als Knoten eines Baumes	Geschäftsprozesse als Knoten eines Graphen
Knotenassoziationen	nur *include*	alle UML-Assoziationen
beschriebene Abstraktionsniveaus	alle Niveaus vom Hauptziel bis zu den konkreten Teilzielen	ein od. wenige Niveaus (falls „include“ benutzt wird)
Kontrollflüsse	Angabe in den *Plänen* (Pseudocode)	n.v.
Angabe über Häufigkeit und Dauer der Aufgaben	Angabe in den *Plänen* (Pseudocode)	Als ergänzende Angaben in den Geschäftsprozessen (verbal)

Tabelle 1: Vergleich von HTA und Geschäftsprozessen

Aspekt	HTA	Geschäftsprozesse / Geschäftsprozessdiagramm
Angabe über Effektivität, Effizienz und Zufriedenstellung	n.v.	Als ergänzende Angaben in den Geschäftsprozessen (verbal)
Benutzer	Angabe in den Knoten	als spezielle Knoten

Benutzermermale

Die Benutzermerkmale wie beispielsweise Erfahrung mit der Software und mit der Aufgabe können tabellarisch angegeben werden (Balzert 2000).

Arbeitsmittelmerkmale und Merkmale der physischen Umgebung

Die Spezifizierung der Arbeitsmittel und der physischen Umgebung erfolgt im Abschnitt „Technische Produktumgebung“ des Pflichtenheftes (Balzert 2000).

Maße der Gebrauchstauglichkeit

Die Maße der Gebrauchstauglichkeit werden spezifiziert, indem mindestens ein Maß jeweils für Effektivität, Effizienz und Zufriedenstellung beschrieben wird. Diese Beschreibung erfolgt als ergänzende Beschreibung in den Geschäftsprozessen (Balzert 2000). Falls ausschließlich HTA und keine Geschäftsprozesse benutzt werden sollen, können die Maße der Gebrauchstauglichkeit tabellarisch beschrieben werden.

3.2 SOFTWARE-ERGONOMISCHE RICHTLINIEN

In den letzten 20 Jahren sind mehrere GUI-Systeme entstanden, welche für bestimmte Rechnerarchitekturen und Betriebssysteme entwickelt wurden. Dazu zählen unter anderem *Windows* von Microsoft für den PC, Graphical Environment Manager (GEM) von Digital Research für den Atari ST, Aqua für Apples Mac OS oder auch das X-Window-System für die UNIX-Systeme. Obwohl sich die Oberflächen und das Verhalten der verschiedenen GUI-Systeme in der letzten Zeit einander angenähert haben, hat jedes GUI-System eigene Grenzen, welche die Erfüllung der Richtlinien verhindern oder im unterschiedlichem Maße unterstützen können.

In diesem Abschnitt werden die existierenden Richtlinien (Styleguides) für den Entwurf eines GUI in der Hinsicht verglichen und analysiert, die Arbeit des Operateurs im optimalen Maße zu unterstützen. Die Anwendung der Styleguides wird im Rahmen des Implementierungsabschnittes (Abschnitt) dargestellt.

3.2.1 *Alphanummerische Display Elemente*

Die meisten GUI-Systeme benutzen alphanummerische Elemente. Nach (O'Hara et al. 2002) sollten folgende Richtlinien beachtet werden. Die Schriftarten, -formen, -familien und -größen sollten möglichst einheitlich gewählt werden. Die Schriften mit Serifen, Schatten, dreidimensionale Schriften und Schriften mit weiteren grafischen Erweiterungen sollten vermieden werden. Für Labels müssen alle Zeichen Groß- oder Kleinbuchstaben sein. Ansonsten sollte gewöhnliche Großschreibung benutzt werden: das erste Wort und alle Substantive großgeschrieben. Die Schriftgröße sollte zwischen 16 und 24 Bogenminuten liegen (O'Hara et al. 2002). Die Umrechnung in die Punktgröße wird mit Hilfe der Formel 3.1 durchgeführt.

$$
\begin{aligned}
&\mathbf{h}_s^m - \text{Schrifthöhe als Längenmaß (metr. System)}\\
&\mathbf{h}_s^z - \text{Schrifthöhe als Längenmaß (in Zoll)}\\
&\mathbf{h}_s^b - \text{Schrifthöhe als Bogenmaß}\\
&\mathbf{h}_s^p - \text{Schrifthöhe als Längenmaß (in Punkten)}\\
&\mathbf{d} - \text{Abstand zwischen User und Bildschirm}\\
&\mathbf{h}_s^m = \tfrac{6{,}283 \cdot \mathbf{d} \cdot \mathbf{h}_s^b}{21600}\\
&\mathbf{h}_s^z = \tfrac{\mathbf{h}_s^m}{0{,}0254}; \qquad [\mathbf{h}_s^m] = m \text{ (Meter)}\\
&\mathbf{h}_s^p = \mathbf{h}_s^z \cdot 72; \qquad [\mathbf{h}_s^z] = Zoll
\end{aligned}
\tag{3.1}
$$

Bei einem Abstand 70 *cm* zum Bildschirm ergibt sich eine Schriftgröße zwischen 9 und 14 *Pt*.

3.2.2 *Icons*

Im Operateursarbeitsplatz sind Icons zum einen in den benutzten Buttons integriert, zum anderen werden im Ausgabefenster zur Überwachung der Joystickbewegungen die Joystickvisualisierungen mit jeweils einem entsprechenden Icon versehen, welches MWB repräsentiert. Für die Buttons gilt, dass die Icons die Aktionen, Prozesse und Objekte repräsentieren sollen, welche die Buttons steuern.

Folgende Richtlinien sollten bei dem Design der Icons beachtet werden. Die Icongrafiken sollen so einfach wie möglich sein. Der primäre Vorteil von Icons sollte die schnelle Erkennbarkeit der dahinter liegenden Funktionen sein. Bei der Betätigung der Icons sollen diese hervorgehoben werden (O'Hara et al. 2002). Für den Operateursarbeitsplatz heißt es, dass die Buttons mit den Icons hervorgehoben werden. Durch die Verwendung von grafischen Elementen wird die Reduktion der mentalen Beanspruchung des Operateurs unterstützt (Jessa und M. 2007). Mit Hilfe von Icons wird die Semantik schneller als mit Hilfe einer Beschriftung vom Operateur aufgenommen.

3.2.3 *Farbe*

Die Farben werden im Operateursarbeitsplatz für die Fensterhintergründe, Buttons und für das Fahrobjekt benutzt. Auch der Rahmen zum Hervorheben des Streckenbereiches, welcher für die MWB sichtbar ist, wird farbig hervorgehoben.

Grundsätzlich sollen die Farben konservativ und konsistent benutzt werden. Ist eine Farbe für einen bestimmten Zweck gewählt, sollte keine weitere Farbe für diesen Zweck zusätzlich gewählt werden. Für die Elemente, welche die Aufmerksamkeit des Operateurs benötigen, sollen helle und gesättigte Farben gewählt werden. Wenn Farben zum separieren der GUI-Bereiche oder -Elemente eingesetzt werden, sollen diese Farben klar unterscheidbar sein. Die Abbildung eines Zwecks oder einer Bedeutung auf eine Farbe sollte konform mit der bereits existierenden Farbbedeutung aus dem Tätigkeitsbereich des Operateurs sein. Die Tabelle 2 enthält einen Auszug aus der Beschreibung der Assoziationen der Farben mit bestimmter Bedeutung im Kontext einer Leitwarte in Atomkraftwerken (O'Hara et al. 2002).

Tabelle 2: Assoziationen der Farben in einem Atomkraftwerk

Farbe	Bedeutung	Auffälligkeit
Rot	Gefahr unsicher heiß	gut
Gelb	Risiko Gefährdung Vorsicht anomaler Zustand	gut
Grün	außer Gefahr	mangelhaft

Tabelle 2: Assoziationen der Farben in einem Atomkraftwerk

Farbe	Bedeutung	Auffälligkeit
	befriedigend	
	normaler Zustand	
Weiß	beratend	mangelhaft
	Dampf	

Für die Überwachung von Systemwerten aus dem kritischen Bereich sollte nach der Tabelle 2 die Farbe rot oder gelb gewählt werden.

Das Hervorheben von kritischen Informationen und Zuständen unterstützt Situationsbewusstsein vom Operateursarbeitsplatz (Durso und Sethumadhavan 2008). Der Operateur soll dabei klar und eindeutig zwischen kritischen und weniger kritischen Informationen unterscheiden können (Braseth et al. 2003). Beispielsweise können die Extremwerte der angezeigten Geschwindigkeit durch rote Farbe hervorgehoben werden. Darüber hinaus ist es sinnvoll, unterscheidende Zustände des zu überwachenden Systems und zu unterscheidende Werte von Systemvariablen farbig zu kodieren. Gleichzeitig sollten die statischen Elemente wie beispielsweise Fensterhintergründe eine weniger auffällige Farbe haben. Die hellgraue Farbe eignet sich optimal für diese Zwecke (Burns und Hajdukiewicz 2004).

Bei der Farbwahl der Fenster- und Buttonhintergründe muß allerdings gute Lesbarkeit beachtet werden (O'Hara et al. 2002). In der Tabelle 3 werden einige Farbkombinationen präsentiert, welche eine gute oder schlechte Lesbarkeit unterstützen.

Tabelle 3: Farbenkombinationen für eine gute oder schlechte Lesbarkeit

Farbenkombinationen	Lesbarkeit
Schwarze Schrift auf weißem Hintergrund	sehr gut
Schwarze Schrift auf gelbem Hintergrund	gut
Dunkelblaue Schrift auf weißem Hintergrund	gut
Grasgrüne Schrift auf weißem Hintergrund	gut
Grüne Schrift auf rotem Hintergrund	schlecht
Rote Schrift auf grünem Hintergrund	schlecht
Orange Schrift auf weißem Hintergrund	schlecht

Tabelle 3: Farbenkombinationen für eine gute oder schlechte Lesbarkeit

Farbenkombinationen	Lesbarkeit

3.2.4 *Geometrie der GUI-Elemente*

Für den Operateursarbeitsplatz wurden Entscheidungen über die Größe von Fenstern, Buttons und Schiebereglern getroffen.

Größe

Nach (O'Hara et al. 2002) soll die Anzahl der verschiedenen Größen für ein bestimmtes GUI-Element maximal drei betragen. Dabei sollten die Dimensionen des größten Elementes mindestens als das 1,5-fache des kleinsten definiert werden. Für die Anzahl der verschiedenen Längen eines Elementes ist der maximale Wert gleich sechs. Für alle GUI-Elemente gilt, dass die Dimensionen dieser Elemente der physischen Größe des Bildschirms entsprechen sollen (O'Hara et al. 2002).

Ähnlich wie die Farben zum Kodieren bestimmter Informationen verwendet werden können, kann die Größe eines GUI-Elementes bestimmte Informationen enthalten (Pedersen und Lind 1999). Beispielsweise kann durch die Größe von Buttons die Relevanz oder sogar der Wert der mit dem Button zu kontrollierenden Variable ausgedrückt werden.

Form und Position

Falls Datenwerte durch Größe, Form oder Position grafischer Objekte ausgedrückt werden, soll die Größe oder die Positionsänderung proportional zu Datenwerten festgelegt werden (O'Hara et al. 2002). Eine Interferenz unterschiedlicher Zustände soll vermieden werden. Das heißt, die Einzelnen Zustände sollen klar unterscheidbar sein (Rhodes et al. 2000). Beispielsweise soll der Operateur die angezeigten verschiedenen Joystickpositionen klar unterscheiden können.

Die GUI-Elemente zum Bearbeiten zusammenhängender und zusammengehörender Aufgaben sollen gruppiert werden. Eine solche „zielorientierte" Platzierung von GUI-Elementen unterstützt SB (Endsley 1999). Beispielsweise kann die Geschwindigkeitsüberwachung und die Überwachung von Positionsgenauigkeit des Fahrobjektes in einem GUI-Element integriert werden.

Gleichzeitig ist es sinnvoll, das grafische Interface in Steuerungszone und Überwachungszone zu teilen (O'Hara et al. 2002). Dabei kann jede Zone in einem separaten Fenster integriert werden. Die derartigen Gruppenbildungen können aus einer HTA

abgeleitet werden. Wegen der aufgabenorientierten Natur solcher Gruppenbildungen wird eine Aufgabe schneller bearbeitet als mit einem solchen GUI, in dem die GUI-Elemente zur Bearbeitung einer Aufgabe voneinander geometrisch weit entfernt sind.

Die visuelle Gruppenbildung von zusammengehörenden GUI-Elementen kann durch die Separierung von den erzeugten Gruppen unterstützt werden (O'Hara et al. 2002).

Zum schnellen Erkennen der Funktionalität der GUI-Elemente soll jede Gruppe oder jedes Fenster mit einem Titel versehen werden. Dabei soll der Titel oben mittig ausgerichtet werden und mindestens eine Schrifthöhe von den GUI-Elementen des entsprechenden Fensters separiert werden (Han et al. 2007).

Die Informationsdichte eines GUI ist ein wichtiger Einflussfaktor auf die Effektivität des Operateursarbeitsplatzes und soll daher beachtet werden (Braseth et al. 2003). Die physische Bildschirmgröße soll daher entsprechend gewählt werden (O'Hara et al. 2002).

In (Han et al. 2007) wird darüber hinaus darauf hingewiesen, dass bei komplexen GUIs redundante GUI-Elemente entfernt werden müssen. Dadurch kann der Operateur alle GUI-Steuerelemente schneller erreichen. Beispielsweise sollen keine redundanten Buttons für jeden MWB angelegt werden.

3.2.5 *Interaktivität*

Ein wichtiger Bestandteil eines Operateursarbeitsplatzes ist ein Hilfesystem. Es kann den Operateur insofern entlasten, als es ihn rechtzeitig über anormale Werte des Systems informiert. Beispielsweise kann eine Benachrichtigung an den Operateur gesendet werden, falls der Button „Forward" vom Operateur betätigt wird, während die Geschwindigkeit des Fahrobjektes bereits maximal ist (Pridmore 2007).

In (Pedersen und Lind 1999) wird ebenfalls darauf hingewiesen, dass frühzeitige Benachrichtigung des Operateurs über ein von der Norm abweichendes Verhalten sinnvoll ist. Dabei können unterschiedliche Ausmaße der Abweichung mit Hilfe von Farbe oder auch auditiv kodiert werden (Han et al. 2007). So kann der Operateur handeln, bevor Alarm ausgelöst wird und einen unerwünschten Zustand eventuell vorausschauend verhindern.

Ein interaktives GUI soll den Operateur auch beim Kontrollieren unterstützen (Zhang und et al. 2002). So können beispielsweise Schieberegler nicht nur als GUI-Steuerelement angesehen, sondern zum Ablesen entsprechender Systemwerte verwendet werden. Als Voraussetzung dafür soll allerdings gelten, dass der Wert der gesteuerten Systemvariable nur vom oben genannten Schieberegler verändert werden kann.

In (O'Hara et al. 2002) wird empfohlen, die normalen Zustände mit Hilfe von symmetrischen Formen darzustellen. Für anormale Zustände sollen dagegen asymmetrische Formen benutzt werden.

3.3 GUI-EVALUIERUNG

Eine Evaluierung von GUI kann zum einen begleitend zum Software-Entwicklungsprozess durchgeführt werden. Die Methoden und Werkzeuge kommen bei diesem Ansatz einerseits aus dem Bereich der Informatik (Unterabschnitt 3.3.1), wobei die Evaluierung softwaregestützt durchgeführt wird, andererseits können während des Entwicklungsprozesses Methoden aus der Ingenieurpsychologie, beispielsweise Heuristische Evaluation durch Experten, angewendet werden.

Zum anderen kann eine GUI-Evaluierung empirisch durchgeführt werden. Hierbei kommen Ansätze und Metriken überwiegend aus dem Bereich der Psychologie und Human Factors zum Einsatz (Unterabschnitt 3.3.2). Einer der empirischen Ansätze der Informatik wird im Unterabschnitt 3.3.3 vorgestellt. Voraussetzung für diese Methoden ist ein lauffähiger Prototyp.

3.3.1 *Automatische kriterienbasierte Evaluierung*

Bei einer automatischen kriterienbasierten Evaluierung (AKE) eines GUI wird die formale Spezifikation des GUI auf die Erfüllung der Guidelines toolbasiert validiert. Die Voraussetzung dafür ist eine Formalisierung der zu erfüllenden Guidelines (Moussa und Riahi 2000).

Die Mehrheit der AKE-Werkzeuge bietet allerdings keine Möglichkeit, die Guidelines zu definieren. Die Guidelines sind bei diesen Werkzeugen direkt im Quellcode einkodiert. Zu diesen Werkzeugen zählen AIDE (Sears 1995), A-Promt[1], Bobby[2] und Sherlock (Hamacher 2006). In (Hamacher et al. 2002) wird darüber hinaus ein Werkzeug für die Evaluierung der Dialogkontrolle TREVIS vorgestellt. Auch in diesem Werkzeug sind die Bewertungsregeln fest im Quellcode einprogrammiert (Hamacher und Kraiss 2004).

In (Hamacher und Kraiss 2004) wird ein Konzept zur Evaluierung von GUI und Dialogsystemen anhand der Guidelines vorgestellt. Die grobe Architektur dieses Evaluierungskonzeptes ist in der Abbildung 8 dargestellt. Für die Eingabe der GUI-Beschreibung wird eine objektorientierte Dekomposition verwendet. Diese Beschreibung soll nach dem beschriebenen Konzept automatisch in Fakten transformiert werden. Die Eingabe und Bearbeitung der Guidelines wird dabei mit Hilfe einer regelbasierten Programmiersprache gewährleistet.

1 A-Promt, http://aprompt.snow.utoronto.ca/
2 Bobby, http://jimthatcher.com/bobbyeval.htm

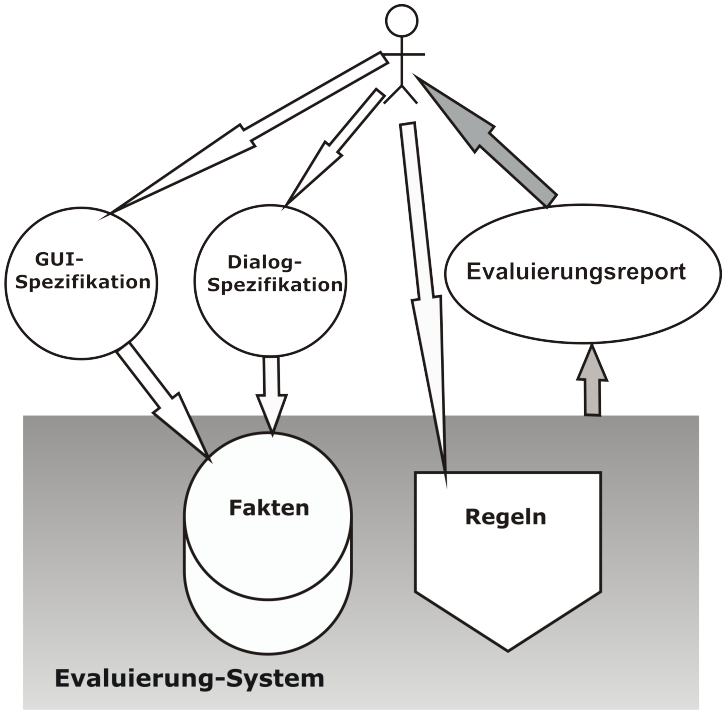

Abbildung 8: Architektur des in (Hamacher und Kraiss 2004) vorgestellten Evaluierungskonzeptes für GUI

Während der Evaluierung werden die Regeln auf die Fakten angewendet. Dabei kann beispielsweise die durch die Regeln definierten Farben, Farbkombinationen oder Überdeckungen von Fenstern überprüft werden. Auch eine Evaluierung der Konsistenz und Angemessenheit der GUI-Ausgaben ist nach dem in (Hamacher und Kraiss 2004) vorgestellten Konzept möglich. Dazu werden Simulationen von Bedienaufgaben durchgeführt.

3.3.2 *Empirische Evaluierung*

In der empirischen oder experimentellen Evaluierung eines GUI wird ein Prototyp von einem Tester bedient und bewertet. Die Bewertung wird nicht zwangsläufig vom Tester abgegeben.

Dabei wird ein GUI indirekt evaluiert, indem SB gemessen wird. Es wird dabei versucht, die Metriken der SB so zu definieren, dass sie nur von Eigenschaften des GUI und nicht von persönlichen Fähigkeiten des Operateurs abhängig sind (Endsley et al. 2003).

Indirekte Metriken

Mit Hilfe von indirekten Metriken wird versucht, anhand bestimmter Maße die SB zu messen. Es sind fünf *indirekte* Metriken zur Messung von SB bekannt (Endsley et al. 2003):

1. Verbale Protokolle: Operateur beschreibt die Bearbeitung der Aufgabe während derer Bearbeitung. Diese Metrik wird auch als *Methode des lauten Denkens* bezeichnet. Sie ist allerdings methodisch nicht sicher, da die Informationsaufnahme durch diese Methode gestört werden kann.

2. Analyse der Kommunikation: nach der Bearbeitung einer Aufgabe wird die Kommunikation zwischen Experten analysiert. Diese Methode wird zum einen von Gedächtniseffekten beeinflüsst, da die Experten nach der Simulation möglicherweise nicht an alle Aspekte der Simulation erinnern können. Zum Anderen ist das Ergebnis dieser Methode davon abhängig, wie standardisiert die Diskussion zwischen Experten abläuft.

3. Verhaltensbasierte Metrik: anhand des Verhaltens und Entscheidungen des Operateurs wird das Niveau von SB bestimmt.

4. performance outcome Metrik: basiert auf der Ausführungsqualität der Aufgabe. Die Ausführungsqualität wird mit Hilfe definierter Standards gemessen, beispielsweise Anzahl der Treffer oder erfolgreicher Missionen. Darüber hinaus spielt bei vielen Messungen die Reaktionszeit des Nutzers eine große Rolle. Dabei wird folgende Hypothese benutzt: je besser SB, desto genauer kann Operateur Aussagen über das zu kontrollierende System und insbesondere über die kritischen Zustände treffen.

Die ersten zwei Typen von Metriken sind von kognitiven Prozessen des Operateurs direkt abhängig. Hier wird versucht, eine direkte Messung von Kognition durchzuführen. Wegen einer starken Abhängigkeit von kognitiven Prozessen des Nutzers sind die Möglichkeiten der ersten zwei Metriktypen zur Evaluation von GUI beschränkt (Endsley et al. 2003).

Um SB mit Hilfe von verhaltensbasierten Metrik zu messen, werden dem Operateur nicht adäquate Systemparameter angezeigt, beispielsweise falsche Flugroute. Gleichzeitig wird die Zeit bis zum Erkennen des Problems vom Operateur und das Lösen des Problems analysiert. Diese Methode ist besonders gut für sicherheitsrelevante Systeme geeignet (Endsley et al. 2003).

Laut (Vidulich 2000) wird in 94% der SB-Studien die performance outcome Metrik verwendet. In 70% dieser Studien war die Performanz der Ausführung der Aufgabe sensitiv zu Änderungen des GUI.

Es muss allerdings beachtet werden, dass keine Abbildung der Performanz auf SB angegeben werden kann. Man kann lediglich ein Wahrscheinlichkeitsmaß im wahrscheinlichkeitstheoretischen Sinne angeben, welches die Performanz auf das Intervall der Wahrscheinlichkeiten $[0, 1]$ abbildet. Mit der Zunahme der Performanz erhöht sich lediglich die Wahrscheinlichkeit dafür, dass

der Wert von SB gut oder sehr gut ist. Es kann aber auch solche Ereignisse im wahrscheinlichkeitstheoretischen Sinne geben, welche gleichzeitiges Auftreten von schlechter Performanz und guter SB und umgekehrt enthalten. Die Wahrscheinlichkeit für solche Ereignisse ist allerdings klein.

Direkte Metriken

Direkte Metriken liefern ohne Zwischenmaß eine Bewertung von SB. Man unterscheidet dabei *subjektive* und *objektive* direkte Metriken (Endsley et al. 2003).

Subjektive Metrik ist durch eine Bewertung des Niveaus von SB definiert. Die Bewertung wird vom Operateur auf der Skala von 1 (niedrich) bis 5 (hoch) abgegeben (Endsley et al. 2003). Damit berichtet der Operateur darüber, wie er seine SB einschätzt.

Die subjektiven direkten Metriken sind für Evaluationen und Vergleiche mit anderen Designs konzipiert. Allerdings sind die Metriken dieser Klasse stark von den Fähigkeiten des Operateurs abhängig (Endsley et al. 2003). Darüber hinaus ist der Korrelationskoeffizient der Bewertung von SB durch den Operateur und der Bewertung von SB durch einen Beobachter mit Expertenwissen sehr klein (Bell und Lyon 2000). Die Nützlichkeit einer solchen Metrik ist daher unzuverlässig und beschränkt.

Die *objektiven* direkten Metriken basieren auf einer Befragung des Operateurs während einer Simulation (Endsley et al. 2003). Dabei kann die Simulation oder der Test von GUI in randomisiert definierten Abständen angehalten und das Nutzerinterface ausgeblendet werden, um den Operateur nach den Aspekten des zu bewertenden Systems zu befragen (Offline-Metrik). Der Vorteil einer solchen Metrik liegt darin, dass durch geeignet gewählte Zeitpunkte der Befragung die Werte über SB nicht so verrauscht wie nach einer Simulation sind. Das Verrauschen der Werte von SB bei einer Evaluation nach der Simulation sind auf die mentale Beanspruchung des Operateurs während der Simulation zurückzuführen. Mit anderen Worten kann ein Nutzer sich besser an die Auffälligkeiten des GUI oder des gesamten Systems während einer Simulation erinnern als nach einer Simulation. Der bekannteste und am häufigsten angewandte Vertreter der Klasse der Offline-Metriken ist *Situation Awareness Global Assessment Technique* (SAGAT).

Es existieren auch Metriken, bei denen die Befragung des Operateurs stattfindet, ohne das zu führende System anzuhalten (Online-Metrik). Dies erlaubt die Dynamik des zu führenden Systems zu erhalten. *Situation Present Assessment Method* (SPAM) ist dabei die bekannteste Online-Metrik.

3.3.3 *Petri-Netz-basiertes Konzept zur Messung der kognitiven Komplexität*

Bei Evaluierung von GUIs durch Messung der kognitiven Komplexität des grafischen Interfaces handelt sich grundsätzlich um einen empirischen Ansatz. In (Rauterberg et al. 1998) wird eine Petri-Netz-basierte Methode beschrieben, welche eine Messung der kognitiven Komplexität eines GUI erlaubt. Zunächst wird der Begriff *Kognitive Komplexität* mit Hilfe der Definition 3.3.1 präzisiert.

Definition 3.3.1 (Kognitive Komplexität) *Unter der kognitiven Komplexität wird der Aufwand verstanden, der vom Menschen zum Verständnis einer Lösung erbracht werden muss (Fenton und Pfleeger 1998).*

Das in (Rauterberg et al. 1998) beschriebene Verfahren benutzt geloggte Eingaben der Nutzer, um schrittweise eine Netzstruktur aufzubauen. Anhand des endgültigen Petri-Netzes, welches die parallel verlaufenden Prozesse der Eingabe, des Feedbacks und der kognitiven Prozesse modelliert, wird mit Hilfe einer Formel das Maß der kognitiven Komplexität bestimmt. Der Wert, welcher mit Hilfe der in (Rauterberg et al. 1998) definierten Formel berechnet wird, kann als *die Anzahl alternativer Entscheidungen* interpretiert werden.

3.3.4 *Bewertung der Ansätze*

Zusammenfassend muss vermerkt werden, dass die empirische Evaluierung durch *verbale Protokolle*, *Analyse der Kommunikation* (indirekte Metriken) und durch subjektive direkte Metriken nicht präzise genug ist. Die objektiven direkten Metriken können allerdings als eine passende Alternative dienen, denn die Abweichung der Evaluierungsergebnisse dieser Metriken von der wirklichen ergonomischen Güte ist anscheinend kleiner als bei den indirekten und subjektiven direkten Metriken.

Der in (Rauterberg et al. 1998) beschriebene Ansatz kann zwar ebenfalls erst nach der Fertigstellung eines Prototypen eingesetzt werden, bietet aber eine formale und präzise Methode zum Messen der Qualität von GUI. Das besondere an diesem Ansatz ist die Abbildung des Lernprozesses auf das Petri-Netz.

Der in (Hamacher und Kraiss 2004) vorgestellte Ansatz erlaubt eine GUI-Evaluierung während der Softwareentwicklung, was nicht nur die Entwicklungszeit und -kosten reduzieren kann, sondern auch mehr Möglichkeit zur Verbesserung der Gebrauchstauglichkeit der Software bietet. Weiterhin wird nach diesem Konzept für die Eingabe der GUI-Beschreibung eine objektorientierte Dekomposition verwendet. Diese Beschreibung kann bei der Programmierung einer GUI die Entwicklungszeit verkürzen,

da in diesem Fall nur wenige syntaktische Anpassungen notwendig wären, um die vorhandene objektorientierte Beschreibung in eine objektorientierte Sprache zu transformieren.

3.4 FORMALE DEFINITION DER GUI-ELEMENTE

Im Rahmen der vorliegenden Arbeit wurde eine Methode zur formalen Beschreibung von GUI-Elementen in Form einer objektorientirten Dekomposition entwickelt.

Sie kann einerseits als eine Methode zum Spezifizieren der Styleguides dienen und sowohl präsize als auch verständlich für ein interdisziplinäres Team die GUI-Eigenschaften darstellen.

Andererseits kann die objektorientierte Dekomposition von GUI-Elementen bei der automatischen kriterienbasierten Evaluierungsmethode (Unterabschnitt 3.3.1) verwendet werden. Darüber hinaus wird die Implementierung beschleunigt, wenn ein solches Modell bereits vorliegt, da insbesondere beim Einsatz der Programmiersprache Smalltalk nur wenige syntaktische Transformationen notwendig sind, um die objektorientierte Dekomposition nach Samlltalk zu überführen.

3.4.1 *Definition der Elemente objektorientierter Dekomposition*

Es wird zunächst das Modell zur objektorientierten Dekomposition beschrieben, indem alle seine Elemente definiert werden.

$$
\begin{aligned}
&\mathcal{E}_{GUI} - \text{Menge aller GUI-Elemente}\\
&\mathbf{W}_{GUI} - \text{Menge der GUI-Elemente vom Typ } \textit{Window}\\
&\mathbf{M}_{GUI} - \text{Menge der GUI-Elemente vom Typ } \textit{Menü}\\
&\mathbf{B}_{GUI} - \text{Menge der GUI-Elemente vom Typ } \textit{Button}\\
&\mathbf{D}_{GUI} - \text{Menge der GUI-Elemente vom Typ } \textit{Display}\\
&\mathbf{S}_{GUI} - \text{Menge der GUI-Elemente vom Typ } \textit{Steuerelement}\\
&\mathcal{E}_{GUI} = \{\mathbf{W}_{GUI}, \mathbf{M}_{GUI}, \mathbf{B}_{GUI}, \mathbf{D}_{GUI}, \mathbf{S}_{GUI}\}
\end{aligned}
\tag{3.2}
$$

Die Elemente der Mengen $\mathbf{W}_{GUI}, \mathbf{M}_{GUI}, \mathbf{B}_{GUI}, \mathbf{D}_{GUI}$ und $\mathbf{S}_{GUI}$ haben folgende Signaturen:

```
aWindow(aDigit, aDigit, aColor, setOfElements):=
Window{
```

```
height: aDigit,
width: aDigit,
color: aColor,
subElements: setOfElements
}

aMenu(aString, aColor, aSetOfItems):=
Menu{
name: aString,
color: aColor,
subMenu: aSetOfItems
}

aButton(aDigit, aDigit, aColor, aLabel):=
Button{
width: aDigit,
height: aDigit,
color: aColor,
label: aLabel
}

aDisplay(aDisplayType, aWidth, aHeight, aColor):=
Display{
type: aDisplayType,
width: aWidth,
height: aHeight,
color: aColor,
}

aControl(aControlType, aWidth, aHeight, aColor):=
Control{
type: aControlType,
width: aWidth,
height: aHeight,
color: aColor
}
```

Dabei gilt für die oben definierten Elemente:

$$
\begin{aligned}
&\texttt{aWindow(aDigit, aDigit, aColor, setOfElements)} \in \mathbf{W}_{GUI} \\
&\texttt{aMenu(aString, aColor, aSetOfItems)} \in \mathbf{M}_{GUI} \\
&\texttt{aButton(aDigit, aDigit, aShape, aColor, aLabel)} \in \mathbf{B}_{GUI} \\
&\texttt{aDisplay(aDisplayType, aWidth, aHeight,} \\
&\qquad \texttt{aShape, aColor)} \in \mathbf{D}_{GUI} \\
&\texttt{aControl(aControlType, aWidth, aHeight,} \\
&\qquad \texttt{aShape, aColor)} \in \mathbf{S}_{GUI}
\end{aligned}
\tag{3.3}
$$

$$
\begin{aligned}
&\forall e \in \texttt{setOfElements} : e \in \mathbf{M}_{GUI} \cup \mathbf{B}_{GUI} \cup \mathbf{D}_{GUI} \cup \mathbf{S}_{GUI} \\
&\forall i \in \texttt{aSetOfItems} : i \in \mathbf{M}_{GUI}
\end{aligned}
$$

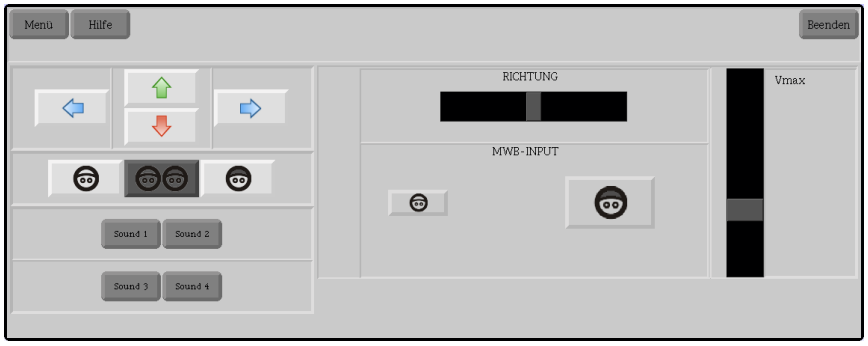

Abbildung 9: GUI-Element controlWindow

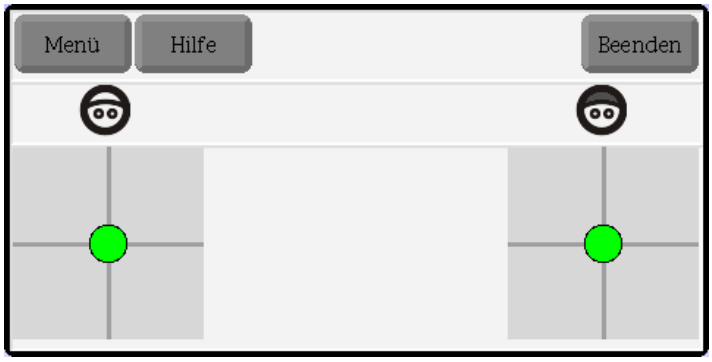

Abbildung 10: GUI-Element joystickOutputWindow

3.4.2 *Objektorientierte Dekomposition von GUI-Elementen des OA*

Die GUI-Elemente von OA wurden mit Hilfe des im Unterabschnitt 3.4.1 definierten Konzeptes beschrieben. Die auf diese Weise entstandene Serialisierung der GUI-Elemente wird im Anhang B dargestellt. Die entsprechende grafische Darstellung der serialisierten GUI-Elemente kann in diesem Unterabschnitt entnommen werden. Dabei werden exemplarisch nur Visualisierungen der Elemente `controlWindow` (Abb. 9), `joystickOutputWindow` (Abb. 10), `mainMenu` (Abb. 11), `leftButton` (Abb. 12), `mwi1UserButton` (Abb. 13), `mwi1InputDisplay` (Abb. 14) und `directionControl` (Abb. 15) gezeigt. Die restlichen GUI-Elemente sind entweder analog gestaltet oder werden im Rahmen der Beschreibung der praktischen Umsetzung im Kapitel 4 vorgestellt und beschrieben. Die Elemente `mwi1InputDisplay` und `mwi1InputDisplay` (*LedMorph*) wurden verworfen und durch das neu entwickelte GUI-Element ersetzt (Unterabschnitt 4.6.6).

Abbildung 11: GUI-Element mainMenu

Abbildung 12: GUI-Element leftButton

Abbildung 13: GUI-Element mwi1UserButton

Abbildung 14: GUI-Element mwi1InputDisplay

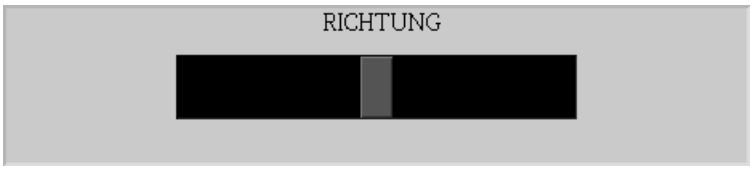

Abbildung 15: GUI-Element directionControl

Im Rahmen der vorliegenden Arbeit wurde einerseits die Softwareentwicklung im Kleinen in Hinsicht auf die Software-Ergonomie untersucht. Andererseits wird in diesem Abschnitt untersucht, inwieweit Vorgehensmodelle die Software-Ergonomie eines Softwareproduktes insbesondere eines GUI beeinflussen.

Der Prozess der Entwicklung eines GUI ist ein evolutionärer Prozess. Einer der Bestandteile dieses Prozesses ist das Interviewen des Auftraggebers oder des Nutzers, um die Anforderungen zu spezifizieren. Auch Testen und Evaluieren von GUI gehört zum Entwicklungsprozess, welcher normalerweise mehrmals iteriert wird, bis das GUI die gewünschte Gestalt und das erwartete Verhalten besitzt (Zhang und et al. 2002).

In (Balzert 2000) wird ein schrittweises Vorgehen für die Entwicklung einer Dialog-Schnittstelle einer Anwendung beschrieben. Dabei ist ein Dialog als eine Interaktion zwischen einem Benutzer und einem System definiert, um ein bestimmtes Ziel zu erreichen. Die folgende Vorgehensweise kann daher für die Entwicklung des GUI des Operateursarbeitsplatz verwendet werden:

1. Skizze der benötigten Fenster und ihrer gegenseitigen Interaktion
2. Definition der Bestandteile eines Fensters
3. Definition der Ereignisbehandlung
4. Entwurf unter Berücksichtigung der benutzten Programmiersprache
5. Programmierung

H. Balzert betont weiterhin in (Balzert 2000), dass „die Normen der Software-Ergonomie sich mit modernen Software-Entwicklungsmethoden und Entwicklungsprozessen wie evolutionäre und inkrementelle Vorgehensweisen in Einklang bringen lassen".

M. R. Endsley beschreibt in (Endsley et al. 2003) das GUI-Design als einen iterativen Prozess. Dieser Prozess wird in der Abbildung 16 grafisch dargestellt. Darüber hinaus wird in (Endsley et al. 2003) auf das *Concurrent Engineering Modell* (CEM) als oft eingesetztes Modell bei GUI-Entwicklung hingewiesen, welches auch als *Simultaneous Engineering Modell* bezeichnet wird.

Das CEM (Abb. 17) ist eine Modifizierung des im Software-Engineering bekannten *Wasserfallmodells*, welche ein gleichzeitiges Bearbeiten und eine Korrespondierung mehrerer Phasen erlaubt. CEM begünstigt die interdisziplinäre Arbeit insofern, dass das Zeitintervall, in dem die Nicht-Softwareentwickler in den Entwicklungsprozess einsteigen können, vergrößert wird.

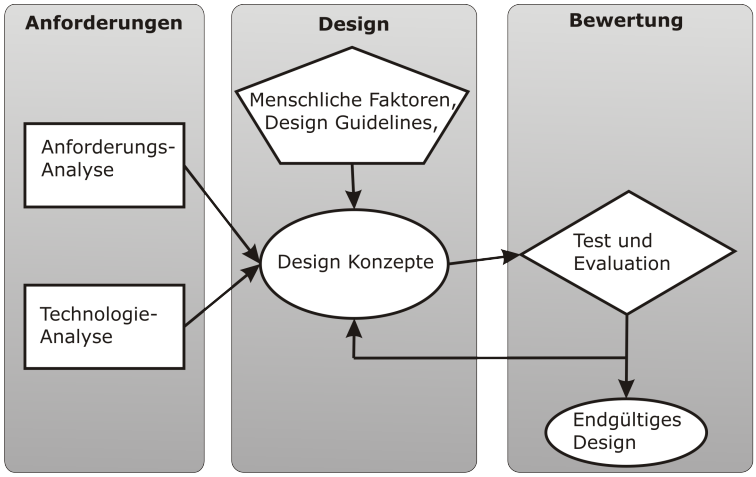

Abbildung 16: GUI-Design nach Endsley

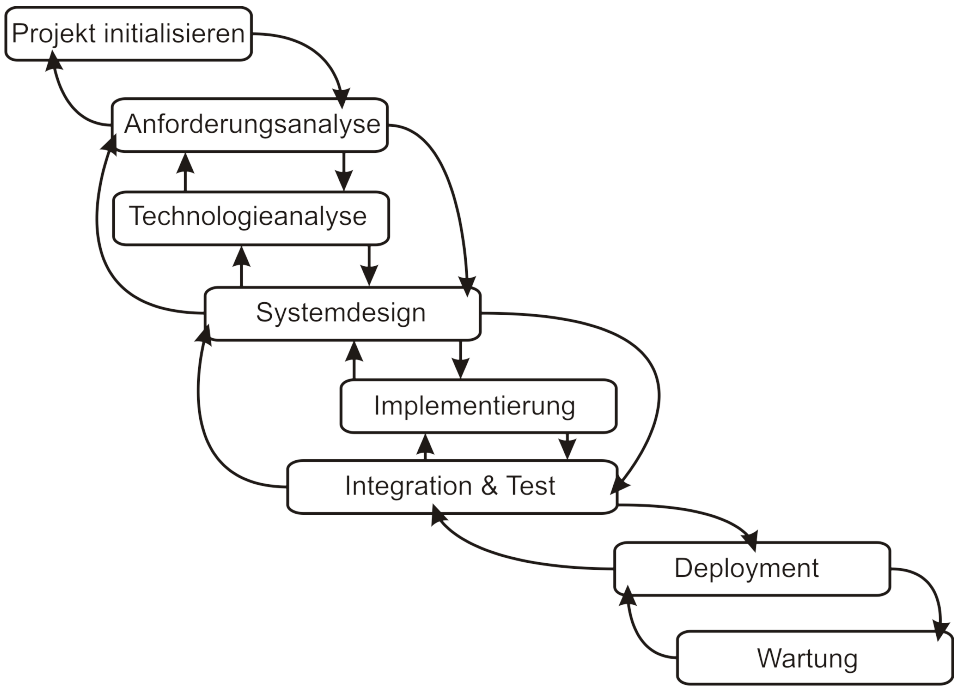

Abbildung 17: Simultaneous Engineering Modell

Ein weiterer Vorteil von CEM ist die Möglichkeit, nicht nur eine zurückliegende Phase zu wiederholen, wie es im Wasserfallmodell üblich ist, sondern es können zwei zurückliegende Phasen iteriert werden. Allerdings muss beachtet werden, dass durch die Iteration die Entwicklungszeit und -kosten auf Kosten der Möglichkeit, die Softwarequalität zu verbessern, steigen können. Hier liegt somit ein Trade-off zwischen der Softwarequalität und Entwicklungszeit (-kosten) vor.

Im Idealfall, bei dem keine Iterationen der Phasen notwendig sind, können *CEM*[3], *Wasserfallmodell*[4], *Spiralmodell*[5] oder *V-Modell*[6] als gleichwertig in der Hinsicht der Optimierung von Softwarequalität und Entwicklungszeit (-kosten) betrachtet werden. In den Fällen, bei denen Iterationen der Softwareentwicklungsphasen notwendig sind, kann CEM eine bessere Optimierung der Zielfunktionen Softwarequalität und Entwicklungszeit (-kosten) bieten (Khalfan und Anumba 1999).

CEM entspricht dem im Gebiet Software Engineering bekannten Treppenmodell (TM). „TM kann dort eingesetzt werden, wo das entwickelte System sinnvoll in ein Kernsystem und darauf basierende Ausbaustufen unterteilt werden kann“ (Ludewig und Lichter). Die im Rahmen der vorliegenden Arbeit entwickelte Software kann als Kernsystem angesehen werden. Als Ausbaustufen gelten die Erweiterungen wie beispielsweise die Vernetzung oder die Schnittstelle zum Überwachen von Beanspruchungsmaßen.

Das entwickelte Kernsystem ist darüber hinaus ein Prototyp, der bereits einen abgeschlossenen Teil des Zielsystems realisiert. Da zum einen bei der Entwicklung von GUI oder speziell von Überwachungssystemen die Faktoren Usability respektive Situationsbewusstsein zentral sind, und zum anderen diese Faktoren an einem Prototypen evaluiert werden können, kann das Prototyping als für die Entwicklung von Sicherheits- und Überwachungssystemen geeignetes Vorgehensmodell angesehen werden.

Ausgehend von den oben genannten Überlegungen, von zahlreichen Vorteilen von CEM für eine nutzerorientierte Softwareentwicklung (Rauterberg et al. 1995) und von der Tatsache, dass die Entwicklung von GUI oft ein iterativer Prozess ist (Abb. 16), werden das CEM (oder Treppenmodell) und Prototyping als eher in Frage kommende Vorgehensmodelle für die Entwicklung von GUI im Allgemeinen und von Sicherheits- und Überwachungssystemen im Speziellen betrachtet.

Nachdem die software-ergonomischen Richtlinien und das software-ergonomie-orientierte Vorgehensmodell für die Entwicklung eines Operateursarbeitsplatzes definiert wurden, wird im Kapitel 4 die praktische Umsetzung des im Rahmen der Diplomarbeit zu entwickelnden Operateursarbeitsplatzes vorgestellt.

3 Simultaneous Engineering, http://de.wikipedia.org/wiki/Simultaneous_Engineering
4 Wasserfallmodell, http://de.wikipedia.org/wiki/Wasserfallmodell
5 Spiralmodell, http://de.wikipedia.org/wiki/Spiralmodell
6 V-Modell, http://de.wikipedia.org/wiki/V-Modell

4 PRAKTISCHE UMSETZUNG

Im Rahmen der Arbeit wurde eine Squeak-Anwendung entwickelt, die das erarbeitete Konzept eines Operateurarbeitsplatzes praktisch umsetzt.

Im Abschnitt wird zunächst die Anforderungsspezifikation der zu entwickelnden Software vorgestellt, da sie als Voraussetzung für die Erstellung eines OOA-Modells gilt. Als Grundlage einer benutzerorientierten Softwareentwicklung wird im Abschnitt die durchgeführte HTA vorgestellt. Im Abschnitt wird die Interaktion des Nutzers mit der Software mit Hilfe eines Geschäftsprozessdiagrammes und die Funktionalität der Software mit Hilfe von Geschäftsprozessen beschrieben. Als nächstes wird im Abschnitt 4.4 das objektorientierte Modell der zu entwickelnden Software beschrieben. Dabei wird die in (Balzert 2000) beschriebene methodische Vorgehensweise zur Erstellung eines OOA-Modells verwendet. Darüber hinaus werden die entwickelten Klassen und Methoden näher beschrieben.

Im Abschnitt 4.5 werden alle Variablen von SAM vorgestellt. Die benötigten Variablen werden hervorgehoben.

Der Operateursarbeitsplatz (OA) soll eine Softwarekomponente einer verteilten Anwendung darstellen, dessen weitere Komponenten SAM und automatische Fahrassistenz (Automatik) sein werden. Die Automatik wird im Rahmen einer weiteren Diplomarbeit von Kai Kesselring entwickelt. Die bereits bestehende Softwarekomponente SAM und die im Rahmen der vorliegenden Arbeit zu implementierende Komponente OA werden vernetzt. Die Vernetzung der beiden Softwarekomponenten SAM und OA ist die Aufgabe einer weiteren Diplomarbeit, welche zeitgleich im Rahmen des Projektes ATEO von Nicolas Niestroj durchgeführt wird. Die Implementierung des OA wird im Abschnitt 4.6 beschrieben. Da bei der praktischen Umsetzung auch Klassenbibliotheken benutzt wurden, wird deren Wahl begründet.

Das Kapitel wird mit der Vorstellung von Testergebnissen der Software und mit der Beschreibung aufgetretender Probleme abgeschlossen (Abschnitt 5.1).

4.1 SYSTEMANALYSE

An dieser Stelle werden die qualitativen und die quantitativen Eigenschaften der zu entwickelnden Software festgelegt. Damit entspricht der Inhalt der folgenden Unterabschnitte zum größten Teil dem Inhalt eines Pflichtenheftes. Die in folgenden Unterab-

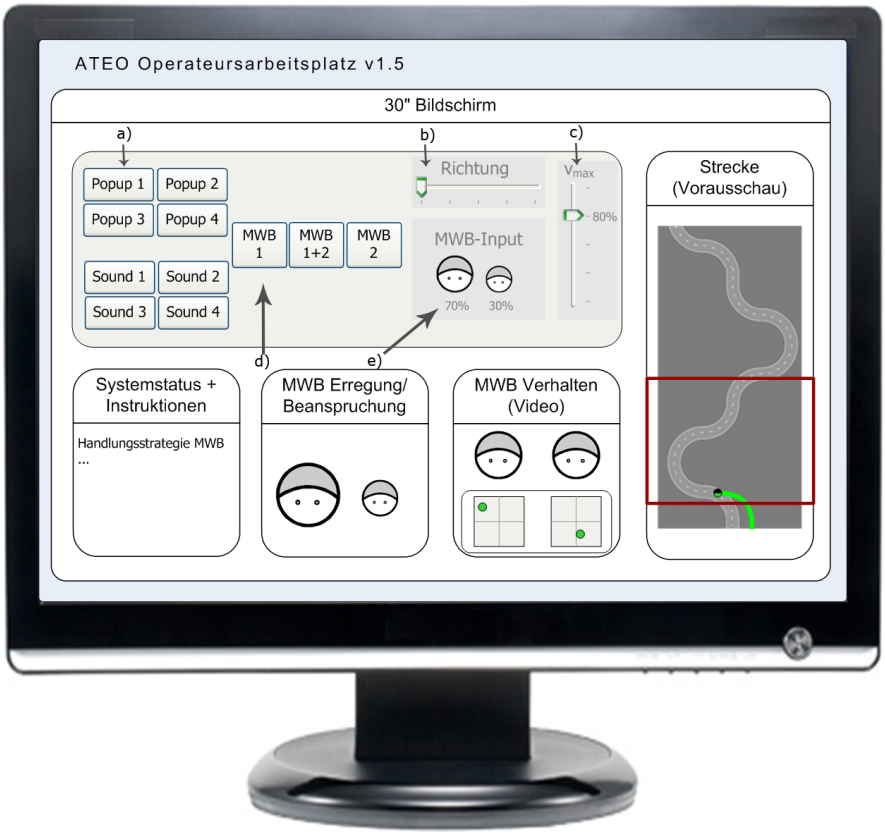

Abbildung 18: Operateursarbeitsplatz v1.5

schnitten durchgeführte Systemanalyse wird als Grundlage für das OOA-Modell verwendet.

4.1.1 *Anforderungen an die Software*

Die Software wurde mit der Smalltalk-Implementierung und Entwicklungsumgebung Squeak entwickelt. Im Rahmen der Diplomarbeit sollen Möglichkeiten des Squeak-Systems im Hinblick auf Überwachung von Echtzeitdaten und Steuerung von SAM untersucht werden. Als Ergebnis dieser Untersuchung soll die vorliegende Software um eine Komponente für den Operateursarbeitsplatz erweitert werden.

4.1.2 *Musskriterien*

Dabei sind folgende Anforderungen an den OA definiert worden.

1. Design und Implementierung von Eingabe/Ausgabe Interface
Eingabe:

- Eingreifen in die Steuerung des Trackingsystems
  - Verstellen des MWB-Inputs, die Summe soll 100% betragen. Nur die Relationen zwischen MWB 1 und 2 sollen verschoben werden können (s. „MWB-Input“, Abb. 18, e))
  - Zwingen des Objekts in eine von zwei Richtungen, für Gabelungen - linke vs. rechte Gabelung (s. Slider „Richtung“, nur 2 Sliderpositionen li. und re., Abb. 18)
  - Beschränken der max.Geschwindigkeit des Objekts (s. Slider „vmax“ in Abb. 18, c))
- Visuelle Anweisungen an die MWBs senden
- Auditive Anweisungen an die MWBs senden

Ausgabe:

- Joystickbewegungen der MWB
- Beanspruchungsmaße (z.B. Herzratenvariablilität, HRV)
- Videoübertragung (Sicht auf die MWB)
- Benutzungsfreundliche Überwachung von Logfilevariablen (s. Abb. 18)
- Überwachung von Leistungsdaten des Objekts/MWB-Teams integriert am Objekt:

  - Genauigkeits/Fehler-History bspw. über einen Schweif
  - Geschwindigkeit (Zeit/Strecke) über die Helligkeit des Schweifs
- Streckenvorausschau mit eingebetteter MWB-Ansicht der Strecke

Das in der Abbildung 18 dargestellte Konzept ist kein endgültiges Design. Es ist eine Studie in der Version 1.5, die durch iteratives Sammeln von Ideen entstanden ist.

2. Editierbarkeit von Ausgabe-/Eingabe Interface

- Editierbarkeit von Ort und Darstellung (Schriftart, Form)
- Möglichkeit, bestimmte Ausgabefenster ein-/auszublenden
- Möglichkeit, das Maß des Eingreifens in die Steuerung frei zu wählen
- Erweiterbarkeit und vereinfachte Ersetzbarkeit visueller/auditiver Anweisungen. Dabei wird das Editieren selbst von einem externen Tool erledigt.

Die Möglichkeit der Editierbarkeit soll mit Hilfe einer Schnittstelle für den Versuchsleiter umgesetzt werden. Diese Möglichkeit stellt eine zentrale Anforderung für die Untersuchungen dar. Darüber hinaus ist eine der zentralen Anforderungen an SAM im Rahmen des Projektes ATEO die Erweiterbarkeit.

4.1.3 *Wunschkriterien*

Es wird vorgesehen, die mentale Beanspruchung der MWB indirekt durch ihre Herzratenvariabilität zu messen. In diesem Zusammenhang soll ein Konzept zur Visualisierung der mentalen Beanspruchung entwickelt werden. Eine praktische Umsetzung des entwickelten Konzeptes wird ebenfalls erwünscht.

4.2 HTA DES OPERATEURSARBEITSPLATZES

In der Anfangsphase der Entwicklung des Operateursarbeitsplatzes wurde eine HTA erstellt, welche die Aufgaben und Ziele des Operateurs wiederspiegelt. Zur Unterstützung der benutzerorientierten Softwareentwicklung wurde die erstellte HTA unter anderem zum Ableiten der Squeak-Klassen benutzt.

Die HTA des Operateursarbeitsplatzes wurde in einem iterativen Prozess erstellt. Eine baum-orientierte Sicht auf die erstellte HTA wird im Anhang A vorgestellt.

4.3 PRODUKTÜBERSICHT

Folgende Beschreibung stellt neben einer Übersicht über die Aufgaben, die durch die Software bewältigt werden, auch die Grundlage für das OOA-Modell dar.

4.3.1 *Beschreibung der Geschäftsprozesse (engl. Use Cases)*

Die Geschäftsprozesse werden aus der bereits durchgeführten HTA abgeleitet und spezifizieren die ergebnisorientierten Arbeitsabläufe bei Benutzung der Software.

/F10/
Geschäftsprozess: GUI Einstellungen vornehmen
Ziel: Anpassung des GUI an die persönlichen Bedürfnisse
Vorbedingung: GUI entspricht nicht den Wünschen des Benutzers
Nachbedingung Erfolg: GUI entsprechend den Wünschen angepasst
Nachbedingung Fehlschlag: keine entsprechende Anpassungsmöglichkeit gefunden
Akteure: Operateur, Versuchsleiter
Beschreibung:
1. Der Operateur prüft die Anwesenheit und Korrektheit der Darstellung von GUI-Elementen
2. Der Operateur lokalisiert das GUI-Menü
3. Der Operateur passt das GUI an
4. Der Operateur überprüft die durchgeführten Änderungen

/F20/
Geschäftsprozess: Systemvariablen in den OA übertragen
Ziel: Ermöglichen einer Überwachung des Systems durch den Operateur
Vorbedingung: Tracking wurde gestartet
Nachbedingung Erfolg: Alle Systemvariablen werden überwacht
Effektivität: Operateur erkennt abnorme Situationen anhand der Ausgaben
Akteure: SAM
Beschreibung:
1. Systemvariablen übertragen
Alternativen:
1a. Daten über Joystickbewegungen übertragen
1b. Daten über mentale Beanspruchung, beispielsweise Herzratenvariabilität (HRV) übertragen
1c. Wert der Geschwindigkeit übertragen
1d. Position des Fahrobjektes übertragen

/F30/
Geschäftsprozess: Indirektes Eingreifen in die SAM-Steuerung
Ziel: Die Anpassung der Systemparameter von SAM mittels MWB erzwingen
Vorbedingung: Tracking wurde gestartet
Nachbedingung Erfolg: Die anzupassenden Systemvariablen wurden von MWB den Wünschen entsprechend angepasst
Effektivität: Operateur kann in das System indirekt jederzeit eingreifen
Akteure: Operateur
Beschreibung:
1. Audiosignal an die MBW senden
Alternativen:
1a. Videosignal an die MBW senden

/F40/
Geschäftsprozess: Direktes Eingreifen in die SAM-Steuerung
Ziel: Systemparameter von SAM anpassen
Vorbedingung: Tracking wurde gestartet
Nachbedingung Erfolg: Die anzupassenden Systemvariablen wurden den Wünschen entsprechend angepasst
Effektivität: Operateur kann in das System direkt jederzeit eingreifen
Effizienz: Reaktionsverzögerung des Systems ist minimal
Akteure: Operateur
Beschreibung:
1. Die Fahrtrichtung an einer Gabelung ändern
Alternativen:
1a. Maximale Geschwindigkeit ändern
1b. Input-Verteilung der MWB ändern

4.3.2 *Geschäftsprozessdiagramm (engl. Use Case Diagramm)*

Die Interaktion der oben beschriebenen Geschäftsprozessen untereinander wird mit Hilfe des folgenden Geschäftsprozessdiagrammes beschrieben.

4.4 OOA- UND OOD MODELL

Als nächstes wird die fachliche Lösung der zu entwickelnden Software mit Hilfe objektorientierter Konzepte wie Klassen, Assoziationen zwischen Klassen, Vererbung, Operationen und Attribute vorgestellt. Im Laufe der Software-Definition wurde sowohl

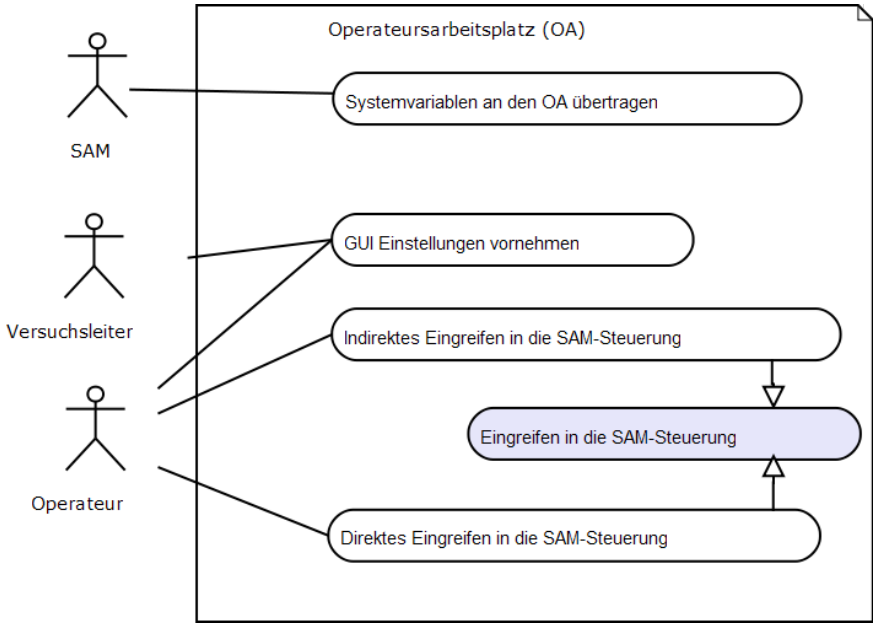

Abbildung 19: Geschäftsprozessdiagramm

ein statisches Softwaremodell als auch eine dynamische Sicht auf die Software erstellt.

Darüber hinaus wurde im Rahmen des Entwurfsprozesses das objektorientierte Designmodell (OOD-Modell) erstellt, indem das objektorientiertes Analysemodell (OOA-Modell) präzisiert und erweitert wurde. Die Definitionsphase und der Entwurfprozess sind zwei chronologisch angeordnete Teile eines evolutionäre Prozesses. Das Ergebnis dieses Prozesses enthält die Beschreibung der Softwarearchitektur und das Klassendiagramm (Anhang C).

Beim Entwurf der Klassen wurde großer Wert auf die Wiederverwendbarkeit gelegt. Da das Projekt ATEO nach Abschluss der vorliegenden Arbeit weitergeführt wird, waren Wartbarkeit und Erweiterbarkeit die Kriterien, welche das Softwaredesign in möglichst hohem Maß erfüllen sollte.

Dabei wurde versucht, die Kohäsion der Klassen zu erhöhen, indem die zusammengehörenden Komponenten in eine Klasse gekapselt wurden. Alle Ein- und Ausgabefenster werden von einer entworfenen Klasse *OAGuiMorph* abgeleitet. Diese GUI-Klasse dient als Template und erlaubt, das Design und das Haupmenü aller Fenster zentral zu entwerfen. Das modulare Design der Buttons *OADistributionButtonMWI1*, *OADistributionButtonMWI2* *OAIconicSwitchButtonLeftUser*, *OAIconicSwitchButtonRightUser* und *OAIconicSwitchButtonBothUsers* entstand nicht nur aus dem Wunsch der höheren Kohäsion, sondern auch aus der Notwendigkeit, auf jede dieser Klassen aus anderen Klassen heraus zuzugreifen, ohne Instanzen dieser Klassen zu besitzen.

Das endgültige Softwaredesign wird im Anhang C vorgestellt. Die entwickelten Klassen und Methoden dieser Klassen werden im Anhang D beschrieben.

4.5 SYSTEMVARIABLEN

Werte aller Systemvariablen von SAM werden äquidistant in einem XML-Format geloggt. Eine Teilmenge dieser Variablen wird im Operateursarbeitsplatz benutzt. Diese Teilmenge wird in der Tabelle 4 angegeben, indem alle XML-Elemente angegeben werden, welche die Logeinträge dieser Variablen enthalten.

In der Tabelle 4 werden ebenfalls die Variablen- und Methodennamen angegeben, deren Werte in den entsprechenden XML-Elementen geloggt werden.

Sowohl die Bezeichnungen der XML-Elemente als auch die Variablen- und Methodennamen beziehen sich auf die Version 1.0 von SAM.

Beschreibung der Variable	Logfilebezeichnung	SAM-Variable/-Methode
Anteil des MWB 1 an der Steuerung	ShareMI1	AnteilMB1
Anteil des MWB 2 an der Steuerung	ShareMI2	AnteilMB2
Schrittweite des Fahrobjektes (Geschwindigkeit)	yDistperTick	schritt
x-Koordinate des Fahrobjekt	PositionX	posX
Links/Rechts-Auslenkung vom Joystick des MWB 1	LeftRightMI1	leftright1
Links/Rechts-Auslenkung vom Joystick des MWB 2	LeftRightMI1	leftright2
Hinweisbild an die MWB	Popup	EventPopup
Sprachbefehl an die MWB	Sound	EventSound
Empfänger des Sprachbefehls (MWB 1 / MWB 2)	Balance	Balance

Tabelle 4: Die geloggten SAM-Variablen, welche für die Vernetzung verwendet werden.

4.6 IMPLEMENTIERUNG

Bei der praktischen Umsetzung wurden 18 Squeak-Klassen entwickelt, wobei ein Teil von ihnen die Eigenschaften bereits vorhandener Squeak-Klassen erbt. Für solche Klassen wurden neue Methoden und Attribute entwickelt, um diese Klassen an die Anforderungen der Software anzupassen. In diesem Abschnitt werden die neuen Methoden und Attribute beschrieben, die relevant für das Verständnis der Softwarearchitektur und für eine Weiterentwicklung der Software sind.

Im Unterabschnitt 4.6.1 wird die Oberklasse der Fensterklassen von OA vorgestellt. Als nächstes wird im Unterabschnitt 4.6.2 das Eingabeinterface beschrieben. Der Unterabschnitt 4.6.3 beschreibt die uFmgesetzte Anzeige der Joystickbewegungen beider MWB. In Unterabschnitten 4.6.4, 4.6.5 und 4.6.6 wird die Implementierung der GUI-Elemente *Button*, *Schieberegler* und des im Laufe der Arbeit entwickelten GUI-Elementes zum Ändern der MWB-Inputverteilung beschrieben. Der Unterabschnitt 4.6.7 gibt die Methoden an, welche das Einlesen von Icons enthalten, um das Ersetzen von Icons zu vereinfachen. Im Unterabschnitt 4.6.8 folgt eine Untersuchung der Problematik der Fahrobjektpositionierung im OA. Im Abschnitt 4.6.9 wird die Umsetzung der Genauigkeit- und Geschwindigkeitsanzeige vorgestellt. Unterabschnitt 4.6.10 beschreibt die Umsetzung der Streckenvorausschau. In Unterabschnitt 4.6.11 wird die Untersuchung aller Möglichkeiten zur Videoübertragung beschrieben. Schließlich werden im Unterabschnitt 4.6.12 die Methoden angegeben, die in der Vernetzung von SAM und OA beteiligt sind.

Da in OA für die Implementierung der Strecke einzelne SAM-Klassen wiederverwendet werden, wird folgende Schreibweise zum besseren Unterscheiden zwischen SAM- und OA-Klassen verwendet: [*OA* | *SAM*].*Klasse»methode*,
[*OA* | *SAM*].*GlobaleVariable methode*.

4.6.1 *Oberklasse OAGuiMorph*

In *OAGuiMorph* wird die Oberklasse für den Eingabemorph *OASAMControlMorph* und den Ausgabemorph *OAJoysticksOutputMorph* definiert. In *OAGuiMorph* werden Menüs zum Ändern der visuellen Eigenschaften des Eingabe-/Ausgabemorphes implementiert, so dass jeder von *OAGuiMorph* abgeleitete Morph mit Hilfe eines Menüs visuell angepasst werden kann.

4.6.2 *Eingreifen in die Steuerung in OASAMControlMorph*

Alle Eingabeelemente werden innerhalb von *OASAMControlMorph* integriert, welches eine abgeschlossene Einheit am Bildschirm bildet (Abb. 20). Auf diese Weise werden die zusam-

mengehörenden Morphe nach den im Unterabschnitt 3.2.4 beschriebenen Styleguides visuell gruppiert. Die Hintergrundfarbe wurde sowohl für den *OASAMControlMorph* als auch für den *OAJoysticksOutputMorph* (Unterabschnitt 4.6.3) gemäß den im Unterabschnitt vorgestellten Styleguides gewählt.

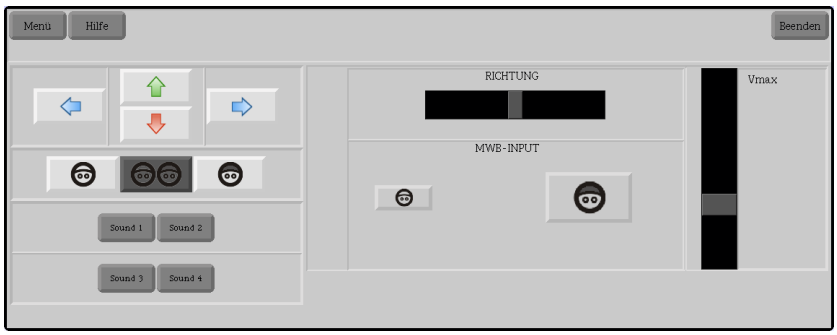

Abbildung 20: *OASAMControlMorph*

4.6.3 *Überwachung der Joystickbewegungen in OAJoysticksOutputMorph*

Während der Steuerung des Fahrobjektes kann es zu Konflikten folgender Art kommen. Der eine MWB kann die Geschwindigkeit des Fahrobjektes erhöhen, während der andere MWB diese Geschwindigkeit drosselt. Analoger Konflikt kann bei der Wahl der Lenkung nach links oder rechts entstehen.

Die Auslenkung des Joysticks wird in den Log-Variablen UpDownMI[1|2] und LeftRightMI[1|2] protokolliert. Anhand der Werte dieser Variablen ist es möglich, das Maß und die Richtung der Auslenkung eines Joysticks abzufragen. Dabei werden die Werte im Intervall $[-1024, 1024]$ benutzt.

Um die Joystickbewegungen der MWB für den Operateur sichtbar zu machen, wurde die Klasse *OAJoystickMorph* implementiert. Dabei handelt sich um ein Morph (Abb. 21). Die relevanten Attribute dieser Klasse sind Instanzvariablen `selfMorphExtent`, `vLine` und `hLine`. Sie werden in der Tabelle 5 beschrieben.

Um die Joystickbewegungen beider MWB zu visualisieren, werden zwei *OAJoystickMorph*-Morphe in einen *OAGuiMorph* integriert (Abb. 22). Diese Integration wird in der Klasse *OAJoysticksOutputMorph* implementiert.

4.6.4 *GUI-Element Button*

Das Senden von auditiven und visuellen Signalen erfolgt über ein Paneel (Abb. 23), welches Buttons zur Auswahl des jeweiligen MWB als auch Buttons zum Senden der Signale enthält. Die Buttons zum Senden von Signalen werden dabei aus der

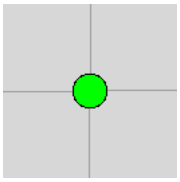

Abbildung 21: *OAJoystickMorph*: der Morph für die Überwachung der Joystickbewegungen

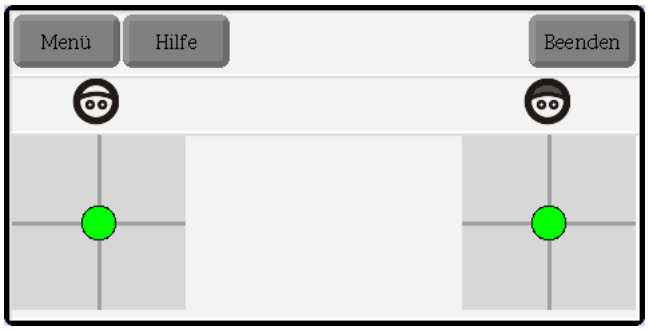

Abbildung 22: *OAJoysticksOutputMorph*: der Morph, in dem die *OAJoystickMorph*-Morphe integriert sind

Klasse *IconicButton* abgeleitet, um eine Grafik auf diesen Buttons platzieren zu können. Die Buttons zum Umschalten zwischen den beiden MWB müssen nicht nur eine Grafik enthalten, sondern ihren Zustand beibehalten können. Squeak bietet die Klasse *SimpleSwitchMorph*, mit deren Hilfe solche Buttons erzeugt werden können, die ihren Zustand beibehalten. Allerdings können solche Buttons keine Grafik enthalten. Da an dieser Stelle eine Mehrfachvererbung sinnvoll wäre, aber keine Unterstützung von Mehrfachvererbungen in Squeak angeboten wird, wurden bei der Implementierung solcher Buttons die Eigenschaften der Klasse *SimpleSwitchMorph* durch Vererbung abgeleitet und die Eigenschaften der Klasse *IconicButton* mit Hilfe der Aggregation hinzugefügt. Bei dem Design aller in OA eingesetzter Buttons wurden darüber hinaus die in den Unterabschnitten 3.2.2 und 4.6.2 beschriebenen Styleguides verwendet.

4.6.5 *GUI-Element Schieberegler*

Für das GUI des OAs werden Schieberegler benutzt. Zum einen wird auf diese Weise die maximale Geschwindigkeit geregelt, zum anderen wird ein Schieberegler benutzt, um das Fahrobjekt in eine bestimmte Richtung zu zwingen. Die erste Version des GUI-Designs enthielt ein GUI-Element zur Ausgabe des Variablenwertes, welcher mit Hilfe der Schieberegler verändert werden konnte. Es wurde ein *LedMorph* als GUI-Ausgabeelement

Instanzvariable der Klasse *OAJoystickMorph*	Beschreibung der Variable
`selfMorphExtent`	Dimensionen des Morphes, in dem die Joystickbewegungen dargestellt werden
`vLine`	Vertikale Linie des Kreuzes im *OAJoystickMorph*
`hLine`	Horizontale Linie des Kreuzes im *OAJoystickMorph*

Tabelle 5: Instanzvariablen der Klasse *OAJoystickMorph*

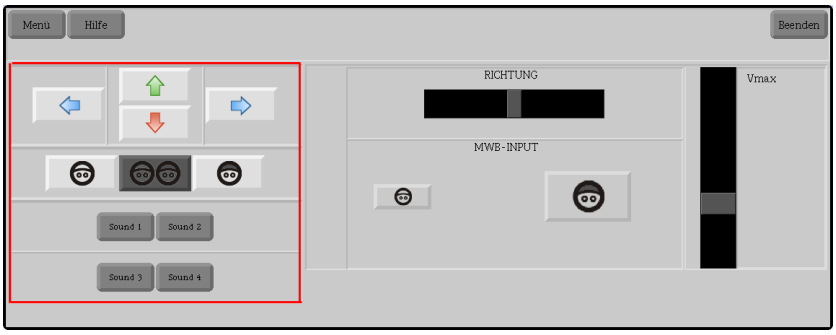

Abbildung 23: *Das Buttonspaneel (rot umrahmt)*

verwendet. Nach den Untersuchungen der Styleguides, welche im Unterabschnitt 3.2.5 vorgestellt werden, wird auf die redundante Ausgabe mit Hilfe von *LedMorphen* verzichtet, um die mentale Beanspruchung des Operateurs in Grenzen zu halten. Die Redundanz der Ausgabe entsteht aus der Möglichkeit eines Schiebereglers zur analogen Anzeige des zu steuernden Wertes.

Für die Implementierung der Schieberegler wurde eine neue Klasse *OASliderMorph* angelegt, die eine abgeleitete Klasse von *SimpleSliderMorph* ist. Die Größe des Schiebereglers beträgt 16 x 100 Pixel (Abb. 24) und kann mit Hilfe der geerbten Instanzmethode *OASliderMorph»extent:* geändert werden (Abb. 25). Die Größe des beweglichen Teils des Schiebereglers beträgt 14 x 7 Pixel. Die Größe des beweglichen Teils kann nur indirekt geändert werden, indem die Größe des gesamten Schiebereglers verändert wird (Abb. 25). Allerdings wird dabei nur die Größe entlang der Achse verändert, welche senkrecht zu der Achse der Bewegung des beweglichen Teiles ist. Die Größe des beweglichen Teils entlang der anderen Achse (Breite) wird dabei nicht verändert (Abb. 25).

Da die Klasse *SimpleSliderMorph* und ihre Oberklasse keine Methode zum direkten Ändern der Breite des beweglichen Teils anbietet, wurde die Instanzmethode *OASliderMorph*»`setSlider-`

Abbildung 24: SimpleSliderMorph im ursprünglichen Zustand

Abbildung 25: SimpleSliderMorph mit geänderter Größe (50 x 200)

`Thickness:` in der Klasse *OASliderMorph* des OAs implementiert. Diese Methode ändert den Wert der neu angelegten Instanzvariable `slTh`, in der die Breite persistiert wird. Anschließend wird in dieser Methode die Initialisierung der Klasse *OASliderMorph* durchgeführt, da dadurch die neue Breite aus der Instanzvariable `slTh` gelesen wird.

Das folgende Listing 4.1 zeigt die Weise zum Ändern der Größe des Schiebereglers (Zeile 2) und des beweglichen Teils (Zeile 3). Der auf diese Weise erzeugte Schieberegler ist in der Abbildung 26 dargestellt.

Abbildung 26: SimpleSliderMorph mit geänderter Größe des beweglichen Teils (Breite = 25)

```
1 slider := OASliderMorph new.
2 slider setSliderThickness: 25.
3 slider extent: 50@200.
4 slider openInWorld.
```

Listing 4.1: Ändern der Größe eines Schiebereglers

4.6.6 *GUI-Element zum Ändern der Verteilung von MWB-Input*

Im Rahmen der vorliegenden Arbeit wurde ein neues GUI-Steuerelement entwickelt, welches die im Unterabschnitt 3.2.4 beschriebenen Styleguides erfüllt. Dieses Steuerelement kann durch seine Größe den Wert der mit dem Button zu kontrollierenden Variable ausdrücken (Abb. 27, 28 und 29). Es besteht aus zwei Buttons (Instanzen von *OADistributionButtonMWI1* und *OADistributionButtonMWI2*), welche als Ereignisbehandlung nicht nur die Verteilung des MWB-Inputs, sondern auch die Größe der Buttons entsprechend dieser Verteilung ändern.

Insgesamt wurden 5 Zustände der Verteilung definiert: (0; 1), (0,25; 0,75), (0,5; 0,5), (0,75; 0,25), (1;0). Somit kann jeder dieser zwei Buttons einen der 5 Zustände annehmen. Die aktuellen Zustände beider Buttons werden in der Klassenvariable *state* der Klassen *OADistributionButtonMWI1* und *OADistributionButtonMWI2* gespeichert. Alle möglichen Zustandswerte, Ereignisbehandlungen und die Größen dieser Buttons werden in den Methoden *OAInputMorph»distributionMWI1ButtonAction* und *OAInputMorph»distributionMWI1ButtonAction* definiert.

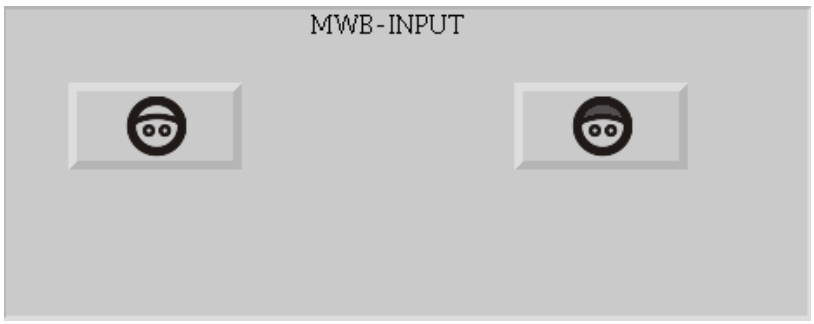

Abbildung 27: GUI-Element zum Ändern der Verteilung von MWB-Input, Zustand: *normal-normal*

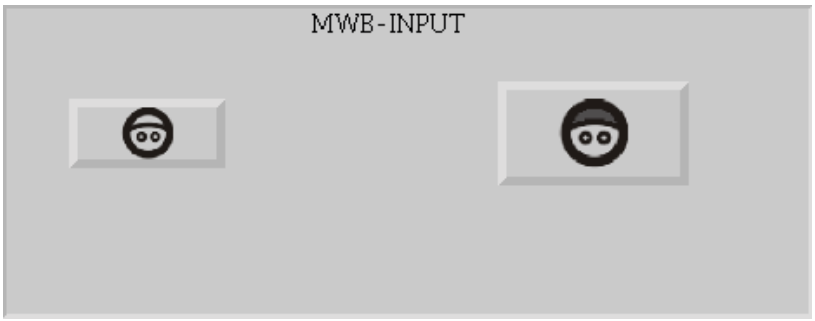

Abbildung 28: GUI-Element zum Ändern der Verteilung von MWB-Input, Zustand: *small-big*

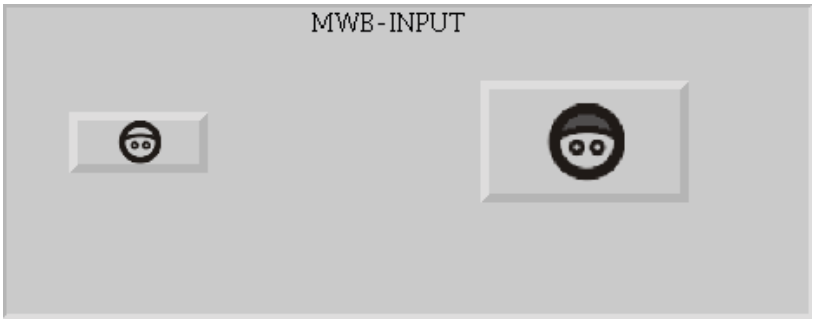

Abbildung 29: GUI-Element zum Ändern der Verteilung von MWB-Input, Zustand: *supersmall-superbig*

4.6.7 *Icons*

Beim Design der Icons wurden die ausgearbeiteten und im Unterabschnitt 3.2.2 vorgestellten Richtlinien verwendet.

In der Tabelle 6 werden alle Methoden zusammengefasst in welchen Icons eingelesen werden. Das Ersetzen von Icons wird mit dieser Tabelle vereinfacht.

Ort des Icons	Methode
Buttons zum Ändern des Inputanteils (*leftuser.png*, *rightuser.png*)	*OAInputMorph*»`makeDistributionButtonsBar`, *OAInputMorph*»`distributionMWI1ButtonAction`, *OAInputMorph*»`distributionMWI2ButtonAction`
Links-, Rechts-, Schneller-, Langsamer-Buttons (*blueleft.png*, *blueright.png*, *greenup.png*, *reddown.png*)	*OAButtonsMorph*»`make[Left|Right|Forward|Back]Button`
Buttons zum Umschalten zwischen den Usern (*leftuser.png*, *rightuser.png*, *bothusers.png*)	*OAButtonsMorph*»`make[Left|Right]UserButton`, *OAButtonsMorph*»`makeBothUsersButton`
Interface zur Überwachung der Joystickbewegungen (*leftuser.png*, *rightuser.png*)	*OAJoysticksOutputMorph*»`make[Left|Right]UserIcon`

Tabelle 6: Die Methoden, welche Icons einlesen

4.6.8 *Überwachung und Kontrolle des Fahrobjektes*

Die Überwachung der Position vom Fahrobjekt auf dem Bildschirm des Operateurs soll möglichst aktuell sein. Die Position des Fahrobjektes wird ursprünglich in SAM definiert, indem die MWB das Fahrobjekt führen und damit die Position des Fahrobjektes ändern. Die Position muss mit Hilfe der Vernetzung zum Operateursarbeitsplatz (OA) übertragen werden.

Es bieten sich zwei Ansätze an, um die Postition des Fahrobjektes auf dem Bildschirm des Operateurs mit der Position des Fahrobjektes auf dem Bildschirm der MWB zu synchronisieren.

Zum einen kann eine zeitbasierte Synchronisation angewendet werden. Dabei fragt eine Methode auf der Seite des OAs zeitlich äquidistant die Position des Fahrobjektes in SAM ab.

Ein anderer Ansatz ist die ereignisbasierte Steuerung. Die möglichen Ereignisse sind dabei die Steuerungseingaben der MWB über die Tastatur oder den Joystick. Bei diesem Ansatz führen die MWB nicht nur das Fahrobjekt des SAM auf eigenem Rechner, sondern auch das Fahrobjekt des OAs.

Es wird der erste Ansatz für die Implementierung gewählt. Die Vernetzung erfolgt dabei mit Hilfe von SOAP. Dabei fungiert der OA als SOAP-Server, falls Systemparameter von SAM auf dem OA angezeigt werden. Will man aber in die Steuerung von SAM eingreifen, wird der SOAP-Server auf der Seite von SAM implementiert. So wird die Methode zum Ändern der Position des Fahrobjektes als SOAP-Service in OA definiert. Die Methoden des SOAP-Clienten werden in SAM definiert und genau dann aufgerufen, wenn die Methoden zur Steuerung des Fahrobjektes aufgerufen werden.

Alle weiteren Aspekte der Vernetzung werden in der zeitgleich durchgeführten Diplomarbeit von Nicolas Niestroj vorgestellt.

Das Fahrobjekt wird in der Methode *OA.ATEOCarNoFb*»`trackerbauen` erzeugt und in der Instanzvariable `carName` der Klasse *ATEOCarNoFb* gespeichert. Global kann man auf das Fahrobjekt über die globale Variable *OA*.`CarID` zugreifen, welche eine Instanz der Klasse *OA.ATEOCarNoFb* ist.

Die Startposition des Fahrobjektes wird in OA analog zu SAM in *OA.ATEOVuSt*»`nextStep` festgelegt (Listing 4.2).

```
1 currentCarCoordinates := OASAMData getCarCoordinates.
2
3 (CarID:=ATEOCarNoFb new) trackerbauen.
4 CarID startpos: currentCarCoordinates.
```

Listing 4.2: Definieren der Startposition vom Fahrobjekt

Die Position des Fahrobjektes zur Laufzeit des Trackings wird in der Methode *OA.ATEOrealJoysticksStepping*»`step` (falls die Inputmethode auf „Joystick“ gesetzt ist) oder in *OA.ATEOAblauf steuerung*»`step` (falls die Inputmethode auf „Tastatur“ gesetzt ist) definiert. Dies wird mit Hilfe der Methode `step: punkt:` durchgeführt (Listing 4.3). Diese Methode definiert darüber hinaus auch die aktuelle Geschwindigkeit des Fahrobjektes (Unterabschnitt 4.6.9)

```
1 currentCarCoordinates := OASAMData getCarCoordinates.
2
3 "Inputmethode: 'Joystick'"
4 Ctrl step:currentSpeed punkt: currentCarCoordinates.
5
6 "Inputmethode: 'Tastatur'"
7 "self step:currentSpeed punkt: currentCarCoordinates."
```

Listing 4.3: Definieren der Position des Fahrobjektes zur Laufzeit

4.6.9 *Geschwindigkeits- und Genauigkeitsüberwachung*

Die Geschwindigkeit des Fahrobjektes wird integriert am Fahrobjekt mit Hilfe von Farben dargestellt. Die minimale Geschwindigkeit wird durch die Farbe Grün ($Farbwert_{RGB} = (0,1,0)_{Dez}$), die maximale Geschwindigkeit durch die Farbe Schwarz ($Farbwert_{RGB} = (0,0,0)_{Dez}$) angezeigt. Die Geschwindigkeiten im Intervall zwischen der maximalen und minimalen Geschwindigkeiten werden mit Mischfarben ausgedrückt. Alle Farben für die Geschwindigkeitsüberwachung werden in der Methode *OASAMData*»`getSpeedColor` definiert (Listing 4.4).

```
1 colors := Dictionary new.
2  colors at:1 put:[Color r:0 g:1 b:0]. "very very slowly"
3  colors at:2 put:[Color r:0 g:0.84 b:0]. "very slowly"
4  colors at:3 put:[Color r:0 g:0.67 b:0]. "slowly"
5  colors at:4 put:[Color r:0 g:0.5 b:0]. "normal"
6  colors at:5 put:[Color r:0 g:0.34 b:0]. "fast"
7  colors at:6 put:[Color r:0 g:0.17 b:0]. "very fast"
8  colors at:7 put:[Color r:0 g:0 b:0]. "very very fast"
```

Listing 4.4: Definieren der Farben für die Geschwindigkeitsüberwachung

Zu den Anforderungen an die zu entwickelnde Software zählt weiterhin eine retrospektive Überwachung der Fahrstrecke des Fahrobjektes. Diese Überwachung wird mit Hilfe eines Schweifes umgesetzt (Abb. 30). Der Schweif zeigt sowohl die kürzlich gefahrene Trajektorie als auch die Geschwindigkeit des Fahrobjektes in sehr naher Vergangenheit.

Die Implementierung des Schweifes wird im Abschnitt E.1 erläutert.

4.6.10 *Strecke*

Für die Erzeugung der Streckenvorausschau werden die existierenden Klassen von SAM benutzt. Sie werden erweitert, um einerseits die Anzeige der Strecke so zu verändern, dass ein längerer Bereich der Strecke sichtbar ist. Andererseits wird die Klasse erweitert, in der die Generierung des Fahrobjektes definiert ist. Die Erweiterung definiert eine integrierte Anzeige der Geschwindigkeit des Fahrobjektes direkt am Fahrobjekt.

Synchronisierung

Um eine synchrone Anzeige der Strecken am Arbeitsplatz der MWB und am Operateursarbeitsplatz zu ermöglichen, sind folgende Klassen und Methoden relevant. Das Betätigen des But-

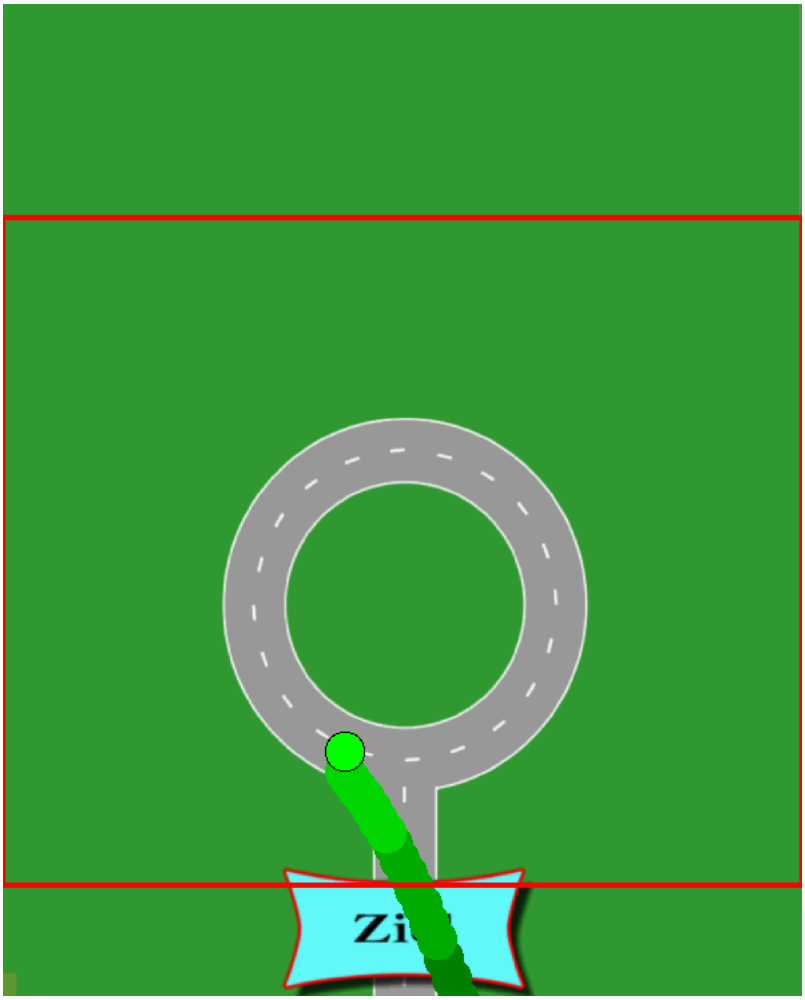

Abbildung 30: Schweif

tons *Los* im Konfigurationsformular von SAM ruft die Methode *SAM.ATEOConfig*»`go` auf. Diese Methode erzeugt eine Instanz der Klasse *SAM.ATEOVuSt* und ruft eine Instanzmethode *SAM.ATEOVuSt*»`start` auf (Listing 4.5). Die Methode *SAM.ATEOVuSt*»`start` bewirkt unter anderem das Einlesen der gesamten Strecke aus den Grafikdateien.

```
1 VuID := ATEOVuSt new.
2 VuID start.
```

Listing 4.5: Einlesen der Strecke nach dem Betätigen des Buttons *Los*

Das Starten des Trackings wird vom MWB durch Betätigen des Buttons „Weiter“ im Fenster mit den Instruktionen veranlasst. Die Ereignisbehandlung dieses Buttons enthält folgende Zeilen:

```
1 lprocess := [
2   self countdown.
3   Ctrl start.
4   ] newProcess.
5 lprocess resume.
```

Listing 4.6: Ereignisbehandlung der Tastenkombination *Alt+S*: Starten des Tracking

Die Variable `Ctrl` ist eine globale Variable und referenziert auf eine Instanz der Klasse *SAM.ATEOAblaufsteuerung*. Demzufolge wird in der Zeile 3 des Listings 4.6 die Methode *ATEOAblaufsteuerung*»`start` aufgerufen.

Um eine synchrone Anzeige der Strecken am Arbeitsplatz der MWB und am Operateursarbeitsplatz zu gewährleisten, sind folgende Schritte notwendig:

- Die Einträge der Datei *steps.txt* müssen in SAM und in OA übereinstimmen.
- Mit dem Betätigen des Buttons *Los* am Bildschirm der MWB soll neben der Methode *SAM.ATEOVuSt*»`start` in SAM als auch *OAControl(class)*»`loadConfigsAndGraphics` in OA aufgerufen werden.
- Mit dem Betätigen des Buttons *Weiter* soll neben der Methode *ATEOAblaufsteuerung*»`start` in SAM auch die Methode *OAControl(class)*»`loadConfigsAndGraphics` in OA aufrufen.

Vorausschau

Die Ansicht der Strecke ist auf eine vordefinierte Pixelanzahl beschränkt. Um die Ansicht eines längeren Streckenabschnittes zu erzeugen, wurden folgende Änderungen untersucht:

- Der Parameter `y` der Nachricht `bottom:y` in der Methode *ATEOPar*»`streckeBauen:` (Anhang E.1, Listing E.1) wird durch einen Parameter `y` > 768 ersetzt.
  Die MWB haben dabei vor dem Starten des Trackings eine Ansicht auf die Strecke, die > 768 Pixel lang ist.

- Beim Umwandeln aller Streckengrafiken in Morphe (Methode *ATEOPar*»`bilderLaden:`, Listing 4.7) werden die Grafiken mit Hilfe der Methode *scaledToSize:* auf kleinere Dimensionen mit unterschiedlichem Faktor skaliert, um einen längeren Streckenabschnitt auf dem Bildschirm zu erzeugen.
  Diese Möglichkeit wird noch nicht für den Produktivbetrieb vorgesehen. Der Grund dafür ist die Komplexität der Geschwindigkeitsberechnung in diesem Fall, da die Geschwindigkeit des Trackings zum Zeitpunkt der Tests zusätzlich durch Performanz beeinflusst wurde.

- Die physische Länge aller Spielbretter wurde vergrößert. Gleichzeitig kann darauf geachtet werden, dass alle Spielbretter gleich lang sind.
  Bisher wurde darauf nicht geachtet, was unterschiedlich lange Aussetzer am oberen und unteren Rand der Strecke hervorruft, wenn die Bildschirmauflösung größer als die Länge der Bilder ist.

```
seqColl do:
  [:each |
    dindex:=(seqColl indexOf: each).
    imageArray at: dindex put: (ImageMorph new image:( (
        Form fromFileNamed: each)  scaledToSize: 600@1440
        ) )
  ].
```

Listing 4.7: Umwandeln aller Streckengrafiken in Morphe

Positionieren der Strecke

Die vorhandene Implementierung der Fahrstrecke, bei der die Kette aus Teilgrafiken (Spielbretter) nach unten bewegt wird, platziert die Grafiken am linken Rand des Bildschirmes. Will man die Strecke mittig oder rechts auf dem Bildschirm positionieren, ist folgende Änderung notwendig.

In der Methode *ATEOPar*»`bilderLaden:` werden Teile der Strecke erzeugt, die sich zum Startzeitpunkt außerhalb des sichtbaren Bereiches befinden. In der letzten Zeile dieser Methode (Anhang E.1, Listing E.3) muss beim Öffnen der Bilder die Nachricht `left:xDelta` mit einem größeren `xDelta` benutzt werden. Dabei ist `xDelta` die Verschiebung der Strecke nach rechts.

Die Methode *ATEOPar*»`streckeBauen:` platziert den Teil der Strecke, der beim Start sichtbar ist. In dieser Methode (Anhang E.1, Listing E.1) sollen die Nachrichten `left:xDelta` mit einem Parameter `xDelta` > 0 versehen werden, um die Strecke nach rechts zu verschieben.

4.6.11 *Videoübertragung*

Als nächstes werden sowohl alle Möglichkeiten des Squeak-Systems als auch alle Klassenbibliotheken zur Unterstützung der Videoübertragung vorgestellt. Mit Hilfe dieser Klassenbibliotheken ist es möglich, ohne großen Aufwand das Video Capturing in den Operateursarbeitsplatz zu integrieren. Vor der Entscheidung, eine bestimmte Klassenbibliothek zur Unterstützung der Videoübertragung zu verwenden, wurden mehrere Ansätze getestet. Die Wahl einer der vorgestellten Möglichkeiten wird begründet.

Zunächst wurde ein Ansatz untersucht, bei dem das übertragende Videosignal nicht innerhalb der Squeakumgebung, sondern in einem Webbrowser erscheint. Eine Squeak-Anwendung kann ebenfalls innerhalb eines Webbrowsers ausgeführt werden (Squeaklet[1]), so dass sowohl die Anwendung als auch das Video innerhalb eines Webbrowser Fensters mit Hilfe von HTML platziert werden kann. Die Squeaklets setzen einen Webserver voraus und werden serverseitig ausgeführt. Auf der Seite des Clients muss ein Squeak-Plugin für den benutzten Webbrowser

1 Squeaklets, http://wiki.squeak.org/squeak/1865

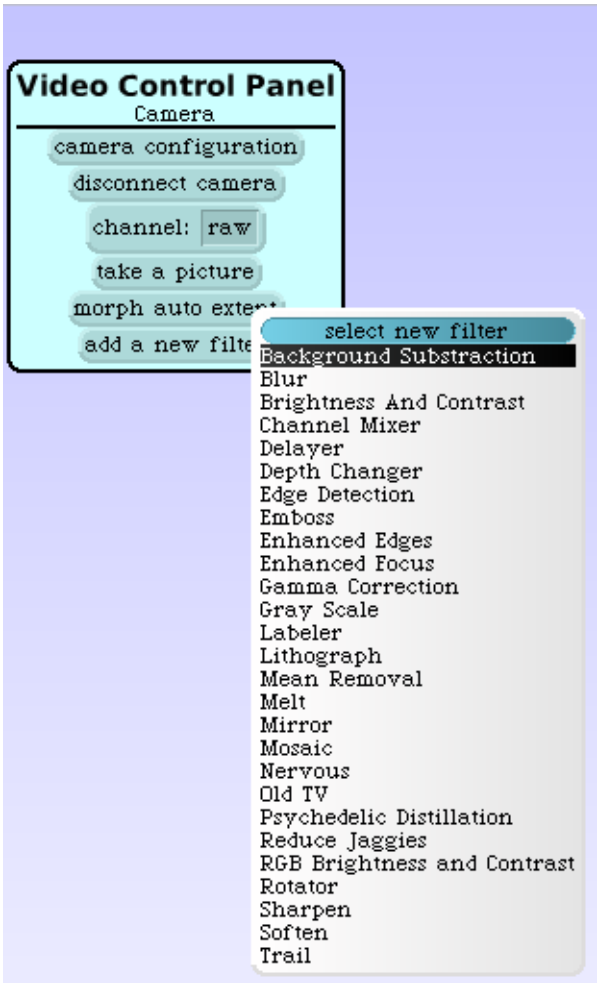

Abbildung 31: Paneel und Filter unter VideoAndImageProcessing

installiert werden. Der Test dieses Ansatzes hat ergeben, dass bei weitem nicht jede Squeak-Anwendung als ein Squeaklet ausgeführt werden kann. Die Anwendungen, die unter Squeak 3.9 entwickelt wurden, konnten nicht gestartet werden. Wenn für die Entwicklung eine frühere Version von Squeak benutzt wurde, liefen einige Anwendungen stabil. Damit eine Squeak-Anwendung als ein Squeaklet garantiert stabil läuft, ist es notwendig die Entwicklungsumgebung aus dem Squeak-Image zu benutzen, das mit dem Squeak-Plugin für Webbrowser mitgeliefert wird. Dieses Squeak-Image bringt aber weitere Nachteile mit sich, beispielsweise eine Inkompatibilität mit den zu installierenden Packages oder das Fehlen bestimmter Klassen, die in den restlichen aktuellen Images vorhanden sind.

Eine weitere Möglichkeit, die MWB per Video zu überwachen, ist eine Darstellung von Bildern innerhalb des in Squeak imple-

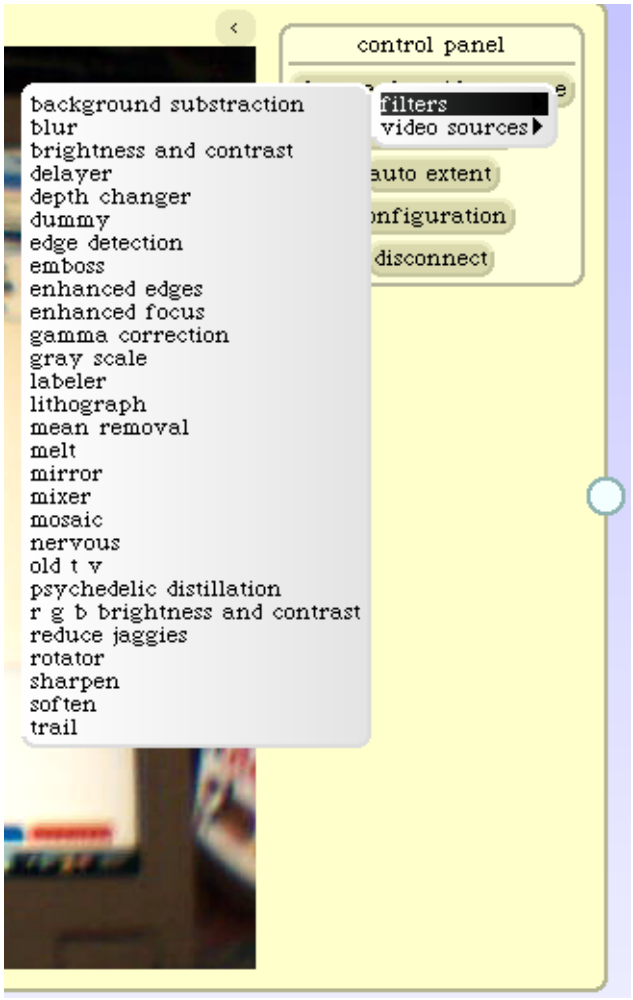

Abbildung 32: Paneel und Filter von VideoFlow

mentierten Scamper-Webbrowsers[2]. Es existieren bisher keine Plug-ins für Scamper, um die Videostreams darzustellen. Um den Scamper-Browser für die Überwachung trotzdem nutzen zu können, müsste eine Webseite erstellt werden, die ein von der Kamera äquidistant zu aktualisierendes Bild enthält. Es sollte ebenfalls äquidistant ein Reload dieser Seite durchgeführt werden, um eine möglichst aktuelle Sicht auf die MWB zu bekommen. Der Reload der Webseite soll in diesem Fall serverseitig durchgeführt werden, da ein clientseitiger Reload den Einsatz von JavaScript voraussetzt. Scamper bietet allerdings bisher keine Unterstützung für JavaScript[3].

Weiterhin wurden Lösungen untersucht, die speziell für das Konvertieren des analogen Videosignals einer Web-Kamera in einen digitalen Videostream (Video Capturing) und zum Anzeigen dieses Streams entwickelt wurden. Insgesamt stehen drei unabhängige Klassenbibliotheken zur Auswahl, die das Video Capturing ermöglichen.

Die Klassenbibliothek WebCam[4] steht als eine ST-Datei zur Verfügung. Der Quellcode muss somit mit Hilfe der FileIn-Funktion ins Squeak-Image geladen werden. Nach dem Laden des Quellcodes erscheint die Klassenbibliothek als eine neue Systemkategorie WebCam-Win32 im Systembrowser. Der Capturing-Prozess wird durch die Anweisung *WebCamMorph new openInWorld* und anschließender Auswahl der Videoquelle gestartet. Über den Halo-Menüpunkt *Menu* kann das Videoformat und die Videokomprimierung geändert werden. Dabei wird die Klassenbibliothek benutzt, die als Dynamic Link Library (DLL) in der Datei sqWebCam.dll mitgeliefert wird.

Zwei weitere Klassenbibliotheken VideoFlow und VideoAndImageProcessing[5] werden als SAR-Packete[6] angeboten. Nach der Installation von VideoAndImageProcessing erscheint im Objektkatalog von Squeak ein neuer Eintrag „Video Morph (from camera)“ unter Menüpunkt Multimedia. Die Installation von VideoFlow bewirkt die Erzeugung eines neuen Menüpunktes Video Flow im Objektkatalog. Dieser Punkt erhält seinerseits das Objekt „Box“.

Die Funktionalität der oben genannten Klassenbibliotheken wurde getestet und verglichen. Der Test wurde auf einer Maschine mit Mobile AMD Sempron Prozessor, Prozessortakt 1,8 GHz, 1024 MB RAM und Windows XP als Betriebssystem durchgeführt. Der Vergleich ist in der Tabelle 7 dargestellt. Die Abbildungen 31 und 32 zeigen die Kontrollpaneels mit dem Untermenü für Filterauswahl von VideoAndImageProcessing und VideoFlow.

2 Scamper Web Browser, http://wiki.squeak.org/squeak/14
3 Scamper. http://wiki.squeak.org/squeak/14
4 WebCam. Autor: Wiebe Baron. http://wiki.squeak.org/squeak/1853
5 VideoFlow, VideoAndImageProcessing. Autoren: Diego Gomez Deck, Alejandro Reimondo, Yoshiki Ohshima, Javier Musa http://wiki.squeak.org/squeak/2411
6 SAR. http://wiki.squeak.org/squeak/3324

	WebCam	VideoAndImageProcessing	Video Flow
Momentaufnahme	über Halo-Menü	integriert	integriert
Kontrollpaneel	kein	über Halo-Menü aufrufbar	integriert
Filter	keine	27 (Abb. 31)	28 (Abb. 32)
Weitere Videoquellen	keine	keine	MPEG-Video
Performanz	ca. 145 $\frac{\text{Frames}}{\text{Min.}}$	ca. 145 $\frac{\text{Frames}}{\text{Min.}}$	ca. 90 $\frac{\text{Frames}}{\text{Min.}}$

Tabelle 7: Vergleich der Funktionalität der Klassenbibliotheken WebCam, VideoAndImageProcessing und Video Flow

Nach einem Vergleich der Funktionalität der Klassenbibliotheken wird VideoAndImageProcessing für den Einsatz im Operateursarbeitsplatz ausgewählt. Die Bibliothek VideoAndImageProcessing besitzt die Schnittmenge der Funktionen, die ausschlaggebend für die Auswahl dieser Bibliothek waren. Es ist nämlich das Vorhandensein eines Kontrollpaneels, integrierte Funktion zur Momentaufnahme und akzeptable Performanz, die sich in der Bildwiederholfrequenz äußert.

4.6.12 *Schnittstellen von OA und SAM für die Vernetzung*

Der Operateursarbeitsplatz und SAM sollen im Rahmen einer weiteren Diplomarbeit von Nicolas Niestroj vernetzt werden. In diesem Unterabschnitt werden die Schnittstellen von OA und SAM beschrieben, um die Implementierung der Vernetzung durchführen zu können. Die Beschreibung der Schnittstellen umfasst eine Angabe von Ereignisbehandlungsmethoden von OA. Weiterhin werden die Klassenvariablen angegeben, welche die von diesen Methoden geänderten Werte enthalten. Die Tabelle 8 enthält die Schnittstellen, welche für die Implementierung der Vernetzung des Eingabeinterfaces des Operateurs mit SAM benötigt werden. Die Schnittstellen für die Vernetzung von SAM und des Ausgabeinterfaces des OAs werden in der Tabelle 9 präsentiert.

Tabelle 8: Schnittstellen der OA-Eingabe

Eingabe-Funktion	OA-Methode (Client)	Klassenvariable mit Wertebereich/-menge
MWB-Input	*OAInputMorph»setDistribution:*	`OAInputMorph(class).mwi[1|2]distributionValue;` $\{0, 0.25, 0.5, 0.75, 1\}$
Richtung	*OAInputMorph»setDirection:*	`OAInputMorph(class).directionValue;` $[0, 100]$
Max. Geschwindigkeit	*OAInputMorph»setSpeed:*	`OAInputMorph(class).speedValue;` $[0, 100]$
Visuelle Anweisungen	*OAButtonsMorph»*[*back*|*forward*|*left*|*right*]*ButtonAction*	
Auditive Anweisungen	*OAButtonsMorph»sound*[1|2|3|4]*ButtonAction*	

Tabelle 9: Schnittstellen der OA-Ausgabe

Ausgabe-Funktion	OA-Methode (Server)	Klassenvariable
Joystickbewegungen	*OAJoysticksOutputMorph(class)»* *[left | right] JoystickTo[Center | East | North | NorthEast | NorthWest | South | SouthEast | SouthWest | West]*	
Position des Fahrobjektes	*OASAMData(class)»setCarCoordinates:*	`OASAMData(class).carCoordinates`
Geschwindigkeit	*OASAMData(class)»setSpeed:*	`OASAMData(class).speed`
Laden der Konfigurationen und Grafiken	*OAControl»loadConfigsAndGraphics*	
Starten des Trackings	*OAControl»startTracking*	

5 ZUSAMMENFASSUNG UND AUSBLICK

In diesem Kapitel werden zunächst die Ergebnisse der Arbeit zusammengefasst. Weiterhin werden aufgetretene Probleme beschrieben und schließlich mögliche Lösungsansätze und Möglichkeiten für eine Weiterentwicklung vorgestellt.

5.1 ZUSAMMENFASSUNG

Mit der vorliegenden Arbeit wurden einerseits Methoden untersucht, welche die SB der von Operateuren überwachten Softwaresysteme erhöhen. Dabei wurden objektive direkte Metriken als effizienteste Methode beurteilt.

Andererseits wurde nach einem effizienten Verfahren gesucht, welches Entwicklung ergonomischer Software, insbesondere Entwicklung von GUI für Operateure zum Überwachen komplexer Systeme, im optimalen Maße unterstützt. Zu den effizienten Verfahren zählt zunächst die geeignete Teilmenge von Styleguides. Die gewählten Styleguides betreffen Vorgaben zu alphanumerischen Displayelementen, Icons, Farben und der Geometrie der GUI-Elemente. Weiterhin zählt zu den herausgearbeiteten effizienten Verfahren ein Konzept zur Evaluierung eines GUI. Die in (Hamacher und Kraiss 2004) vorgestellte formale Evaluierungsmethode wird als sehr effizient und vielversprechend beurteilt. Schließlich spielt bei der Effizienz der GUI-Entwicklung das Vorgehensmodell keine marginale Rolle. Dabei wird die Abschwächung des Wasserfallmodells, das *Concurrent Engineering Modell*, als optimal für die GUI-Entwicklung angesehen.

5.2 PROBLEME, MÖGLICHE LÖSUNGEN UND AUSBLICK

5.2.1 *Rahmen*

Der Rahmen zum Hervorheben des Streckenbereiches, welcher in SAM angezeigt wird, besteht aus vier Linienmorphen. Im Rahmen einer Weiterentwicklung soll der Rahmen als ein einziger Morph implementiert werden. Damit kann möglicherweise die Performance geringfügig verbessert werden. Die Umsetzung kann beispielsweise durch eine Integration von bereits implementierten Linienmorphen in ein durchsichtiges Morph stattfinden. Eine Weitere Möglichkeit ist die Umsetzung mit Hilfe eines Rectangle-Morphes mit transparenter Farbe und einem roten Rand.

5.2.2 *Schweif*

Wie im Abschnitt E.1 gezeigt, werden in jedem Schritt des Trackings Kreise erzeugt und den zwei Spielbrettern, die sich zur Zeit des aktuellen Schrittes im Speicher befinden, als Submorphe hinzugefügt. Da allerdings immer nur eins der Spielbretter befahren wird, ist es zunächst hinreichend, nur einem der Spielbretter den erzeugten Kreis hinzuzufügen. Dazu muss aber entschieden werden können, ob das eine oder das andere Spielbrett momentan befahren wird. Diese Methode fehlt bisher. Sie kann jedoch die Performance verbessern, da mit ihr in jedem Schritt auf die Addition eines Submorphes verzichtet werden kann.

5.2.3 *Inputmanipulation*

Bei der Weiterentwicklung des Operateursarbeitsplatzes soll die Anpassung der Inputmanipulation durchgeführt werden. Die Inputmanipulation soll dabei feiner abgestuft werden: die 25%-Schritte sollen durch 5%-Schritte ersetzt werden.

5.2.4 *Joystickbewegungen*

Falls die Performance der Vernetzung es zulässt, sollen darüber hinaus die Joystickbewegungen feiner dargestellt werden.

5.2.5 *Positionierung von Submorphen*

Die Positionierung von Submorphen und GUI-Steuerelementen wird in OA mit Hilfe von *AlignmentMorphen* umgesetzt. Die Submorphe von *AlignmentMorph* werden geometrisch in Matrixform positioniert. *AlignmentMorph* ist das am häufigsten verwendete Layoutkonzept in Squeak. Eine detaillierte Beschreibung dieses Konzeptes kann beispielsweise in (Maloney 2002) nachgeschlagen werden.

Eine punktgenaue Positionierung innerhalb von *AlignmentMorphen* ist allerdings nicht möglich. Falls bei der Weiterentwicklung von OA eine punktgenaue Positionierung erwünscht wird, muss untersucht werden, ob die Klasse *AlignmentMorph* um die Möglichkeit einer punktgenauen Positionierung erweitert oder ein anderes Layoutkonzept gewählt werden soll.

5.2.6 *Performance*

Das Performanzproblem ist bereits in SAM bekannt. Seit der Entwicklung von OA hat sich dieses Problem verschärft. Dafür sind folgende Gründe verantwortlich.

Einerseits wird die Geschwindigkeit des Trackings durch performancelastige Faktoren sowohl in der SAM- als auch in der OA-Komponente beeinflüsst. Der einflussreichste Faktor ist dabei die Erzeugung und das Bewegen von Morphen. Dieser Einfluss hat eine stochastische Natur, da durch Dynamik der Ausgabe (Spielbretter, Fahrobjekt) der Zustand der erzeugten Morphe regelmäßig unvorhersehbar verändert wird. Daher kann dieser Einfluss schwer berechnet werden, um ihn kompensieren zu können (beispielsweise durch Geschwindigkeitsanpassung). Dieser Einfluss ist allerdings relativ gering und kann zunächst vernachlässigt werden. Die Untersuchungen zur Beseitigung dieses Einflusses werden nichtsdestotrotz im Rahmen des ATEO-Projektes durchgeführt und müssen weiter durchgeführt werden.

Andererseits kann es je nach verwendeter Vernetzungstechnologie zu Performanceeinbussen kommen, deren Einfluss signifikanter ist. Im Laufe der abschließenden Tests von OA wurde in Zusammenarbeit mit Nicolas Niestroj eine vorläufige Vernetzung von OA und SAM implementiert.

Dabei wurden zum einen die Ereignisbehandlungsmethoden der GUI-Steuerelemente von OA mit den entsprechenden Ereignismethoden in SAM mit Hilfe von SOAP verbunden. Der Test verlief erfolgreich.

Zum anderen wurde die Geschwindigkeit des Trackings und die Position des Fahrobjektes mit Hilfe von SOAP von SAM zum OA zeitlich äquidistant und hoch frequentiert übertragen. Die Frequenz der Übertragung soll $>= 20$ Hz (alle $<= 50$ ms die Daten übertragen) betragen, da anderenfalls die Synchronität des Trackings in den SAM- und OA-Komponenten nicht gewährleistet ist. Beim Testen wurde eine nicht akzeptable Performancebeeinträchtigung beobachtet, durch welche die Synchronität des Trackings nicht gewährleistet werden kann. Die Performancebeeinträchtigung kommt durch den Protokollaufbau von SOAP. Daher werden andere Möglichkeiten zu einer hoch frequentierten Übertragung von Geschwindigkeit des Trackings und von Fahrobjektposition untersucht. Diese Untersuchungen werden im Rahmen der Diplomarbeit von Nicolas Niestroj durchgeführt und müssen in der Weiterführung der vorliegenden Arbeit beachtet werden.

Zusammenfassend kann vermerkt werden, dass in weiterführenden Arbeiten der Einfluss der performancelastigen Faktoren minimiert werden soll. Beispielsweise kann untersucht werden, ob *wxSqueak*[1] neben den Werkzeugen für die Implementierung von GUI-Steuerelementen auch eine Möglichkeit zum Erzeugen des Trackings anbietet. *wxSqueak* ist ein Interface zu wxWindows-GUI-Bibliothek (auch wxWidgets) für Squeak. *SqueakGtk*[2] kann ebenfalls unter dem gleichen Aspekt getestet werden. Weiterhin

1 wxSqueak, http://www.wxsqueak.org/
2 SqueakGtk, http://squeakgtk.pbwiki.com/

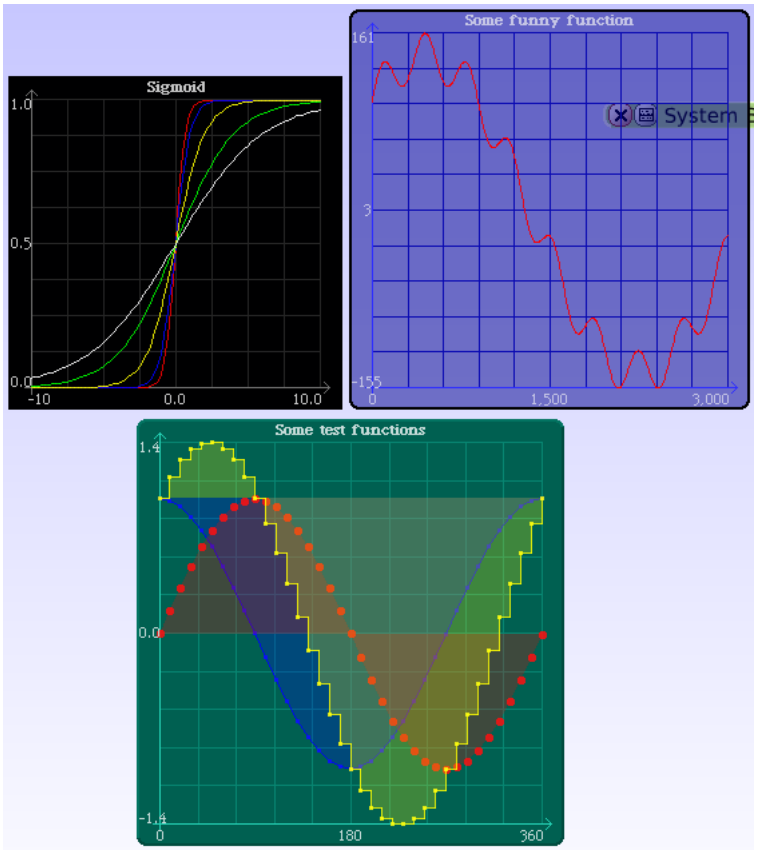

Abbildung 33: PlotMorph Beispiele

kann die JavaScript-Bibliothek *Lively*[3] in Hinsicht auf angebotene Werkzeuge für die Entwicklung des Operateursarbeitsplatzes untersucht werden. *Lively* ist zwar komplett in JavaScript implementiert, benutzt aber den Morphic-Ansatz. Somit können die in Smalltalk implementierten GUIs ohne großen Aufwand nach JavaScript portiert werden, vorausgesetzt, *Lively* bietet vergleichbare Werkzeuge an.

5.2.7 *Beanspruchungsmaße*

Es wird beabsichtigt, die Beanspruchungsmaße wie beispielsweise Herzratenvariabilität (HRV) der MWB zu überwachen. Für die Implementierung einer Visualisierung der HRV kann das Squeak-Package *PlotMorph*[4], welches umfangreiche Möglichkeiten zum Plotten anbietet (Abb. 33), benutzt werden.

3 Lively, http://research.sun.com/projects/lively/
4 PlotMorph, http://wiki.squeak.org/squeak/2626

A

HIERARCHISCHE AUFGABENANALYSE

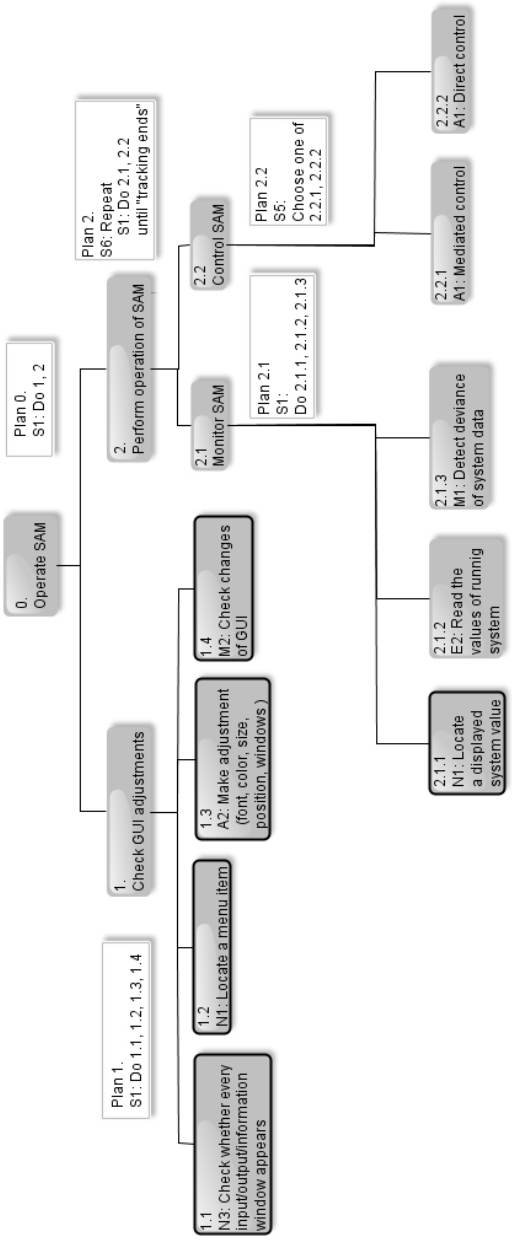

Abbildung 34: HTA Baum bis zur Tiefe 4

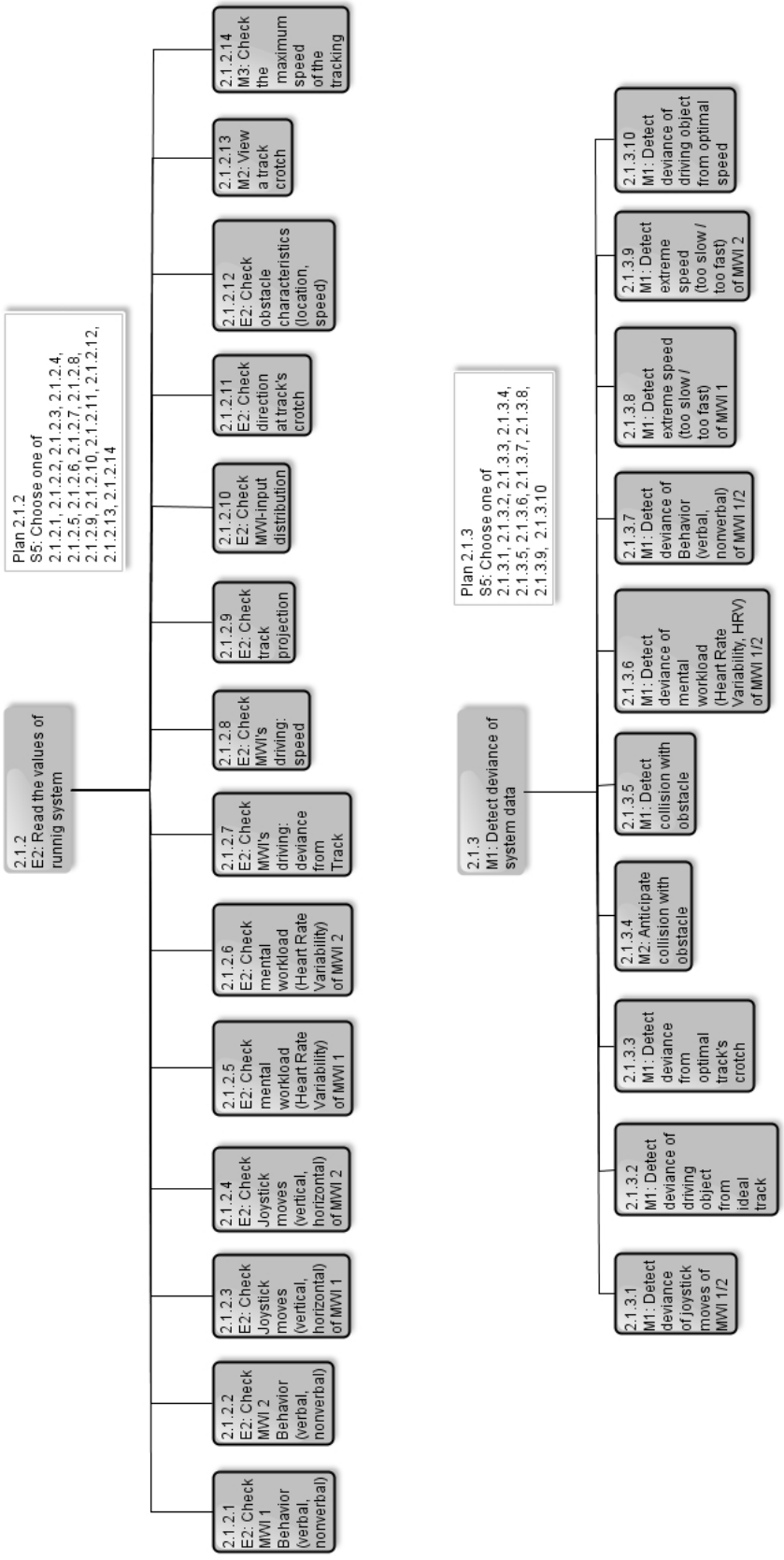

Abbildung 35: HTA, Aufgaben 2.1.2 und 2.1.3

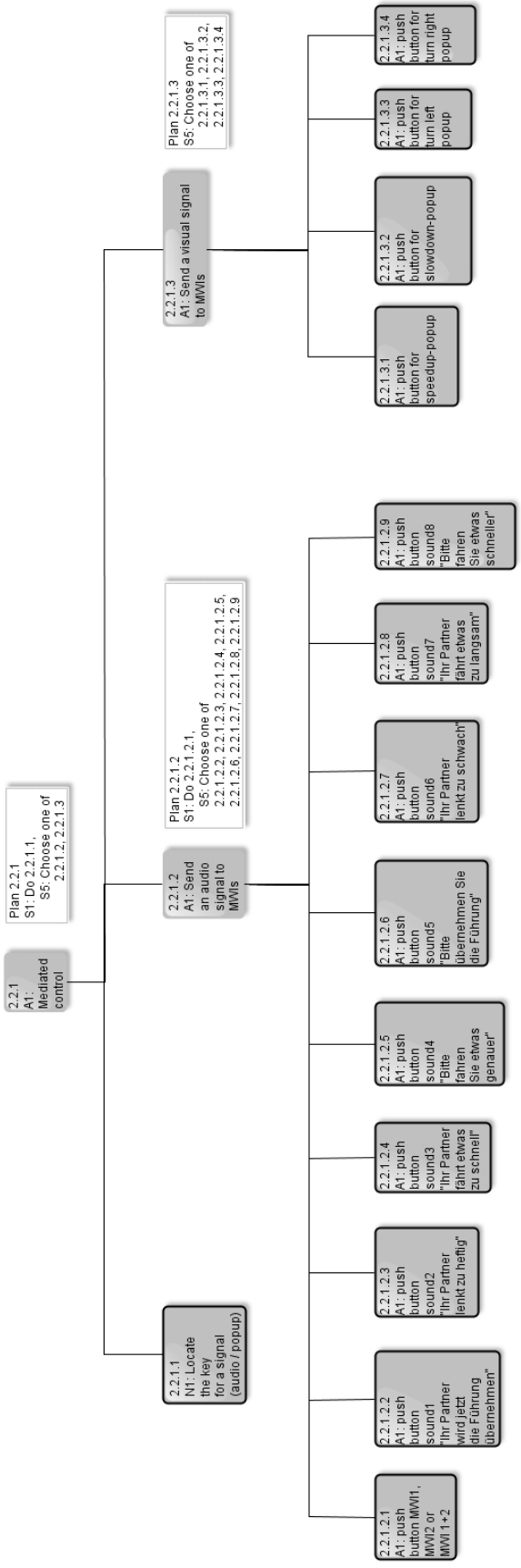

Abbildung 36: HTA, Aufgabe 2.2.1

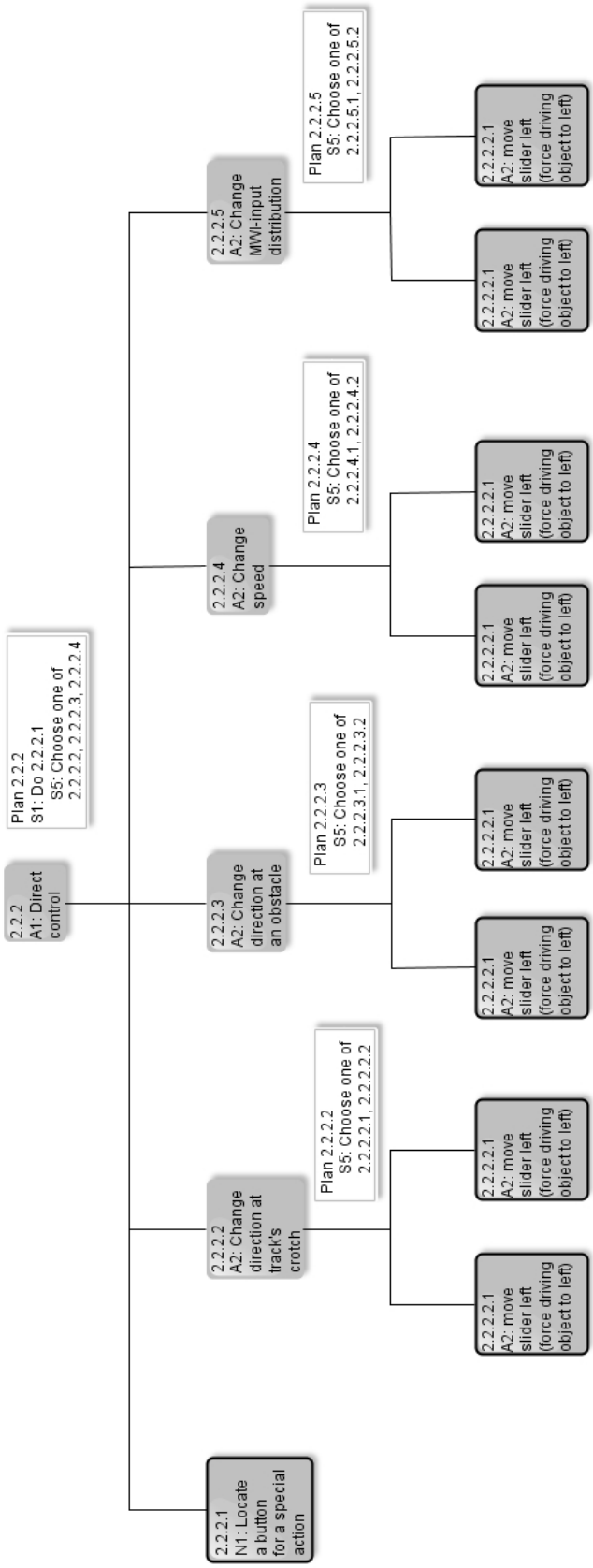

Abbildung 37: HTA, Aufgabe 2.2.2

B

OBJEKTORIENTIERTE DEKOMPOSITION VON OA-GUI

B.1 FENSTER

```
controlWindow := aWindow(380, 1150, gray, {mainMenu, helpMenu,
                 quitMenu, leftButton, rightButton, forwardButton,
                 backButton, mwi1Button, mwi2Button,
                 bothUsersButton, mwi1InputButton, mwi2InputButton,
                 mwi1InputDisplay, mwi2InputDisplay, maxSpeedDisplay,
                 directionControl, maxSpeedControl})

joystickOutputWindow := aWindow(300, 400, gray, {mainMenu, helpMenu,
quitMenu, leftJoystickDisplay, rightJoystickDisplay})
```

B.2 MENÜS

```
mainMenu := aMenu('Menü', gray, {bCMenu, wCMenu, buttonsCMenu,
         ledCMenu, wSMenu, fMenu, helpItem, quitItem})

bCMenu := aMenu('Border Color', gray, {bItem, dgItem, vvdgItem,
gItem, lgItem, vlgItem, vvlgItem, wItem, prItem, lbItem, lmItem})

wCMenu := aMenu('Window Color', gray, {dcItem, prItem, dgItem,
vvdgItem, gItem, lgItem, vlgItem, vvlgItem, wItem, lbItem, lmItem})

buttonsCMenu := aMenu('Buttons Color', gray, {vvlgItem, vlgItem,
lgItem, dgItem, gItem})

ledCMenu := aMenu('LED Display Color', gray, {lgItem, wItem,
oItem, lbItem})

wSMenu := aMenu('Window Size', gray, {smallSizeItem,
middleSizeItem, bigSizeItem})

fMenu := aMenu('Fonts', gray, {smallFontItem,
middleFontItem, bigFontItem})

helpMenu := aMenu('Help', gray, {})

quitMenu := aMenu('Quit', gray, {})

bItem := aMenu('Black', gray, {})
dgItem := aMenu('Dark Gray', gray, {})
vvdgItem := aMenu('Very Very Dark Gray', gray, {})
gItem := aMenu('Gray', gray, {})
lgItem := aMenu('Light Gray', gray, {})
vlgItem := aMenu('Very Light Gray', gray, {})
vvlgItem := aMenu('Very Very LIght Gray', gray, {})
wItem := aMenu('White', gray, {})
prItem := aMenu('Pale Red', gray, {})
lbItem := aMenu('Light Brown', gray, {})
```

```
lmItem := aMenu('Light Magenta', gray, {})
dcItem := aMenu('Default Color', gray, {})
oItem := aMenu('Orange', gray, {})
smallSizeItem := aMenu('Small Size Window', gray, {})
middleSizeItem := aMenu('Middle Size Window', gray, {})
bigSizeItem := aMenu('Big Size Window', gray, {})
smallFontItem := aMenu('Small Font', gray, {})
middleFontItem := aMenu('Middle Font', gray, {})
bigFontItem := aMenu('Big Font', gray, {})
```

B.3 BUTTONS

```
leftButton := aButton(100, 50, Very Light Gray, Left Arrow)
rightButton := aButton(100, 50, Very Light Gray, Right Arrow)
forwardButton := aButton(100, 50, Very Light Gray, Top Arrow)
backButton := aButton(100, 50, Very Light Gray, Bottom Arrow)
mwi1UserButton := aButton(100, 50, Very Light Gray, User 1 Icon)
mwi2UserButton := aButton(100, 50, Very Light Gray, User 2 Icon)
bothUsersButton := aButton(100, 50, Gray, Bottom Arrow)
mwi1InputButton := aButton(100, 50, Light Gray, Bottom Arrow)
mwi2InputButton := aButton(100, 50, Light Gray, Bottom Arrow)
```

B.4 DISPLAYS

```
mwi1InputDisplay := (led, 50, 25, green)
mwi2InputDisplay := (led, 50, 25, green)
maxSpeedDisplay := (led, 50, 25, green)
leftJoystickDisplay := (joystick, 130, 130, green)
rightJoystickDisplay := (joystick, 130, 130, green)
```

B.5 STEUERELEMENTE

```
directionControl := (slider, 250, 20, gray)
maxSpeedControl := (slider, 50, 270, gray)
```

C

SOFTWAREARCHITEKTUR UND KLASSENDIAGRAMM

C.1 SOFTWAREARCHITEKTUR VON OA

Die Abbildung 38 stellt die Softwarearchitektur von OA dar. Die folgenden Unterabschnitte geben die Sequenzen von Schichten in der Reihenfolge an, in welcher die Zugriffe erfolgen, um die angegebenen Aufgaben zu erledigen. Die Pfeile zeigen keine Datenflüsse, sondern Zugriffe einer Schicht auf die andere. Die Daten können dabei in eine sowie in die entgegengesetzte Richtung fließen.

C.1.1 *Überwachung der Joystickbewegungen*

SAM speichert die Daten:
`SAM-JControl` → `SAM-Data Access`

OA liest die Daten ein:
`OA-GUI` → `OA-Business` → `OA-Network` → `SAM-Network` → `SAM-Data Access`

C.1.2 *Überwachung des Trackings*

SAM sendet die Daten an OA:
`SAM-Tracking` → `SAM-Network` → `OA-Network` → `OA-Data Access`

OA liest die Daten ein:
`OA-Tracking` → `OA-Data Access`

C.1.3 *Eingreifen in die Steuerung*

OA aktualisiert die SAM-Daten:
`OA-GUI` → `OA-Business` → `OA-Network` → `SAM-Network` → `SAM-Data Access`

SAM liest aktuelle Daten ein:
`SAM-Tracking` → `SAM-Data Access`

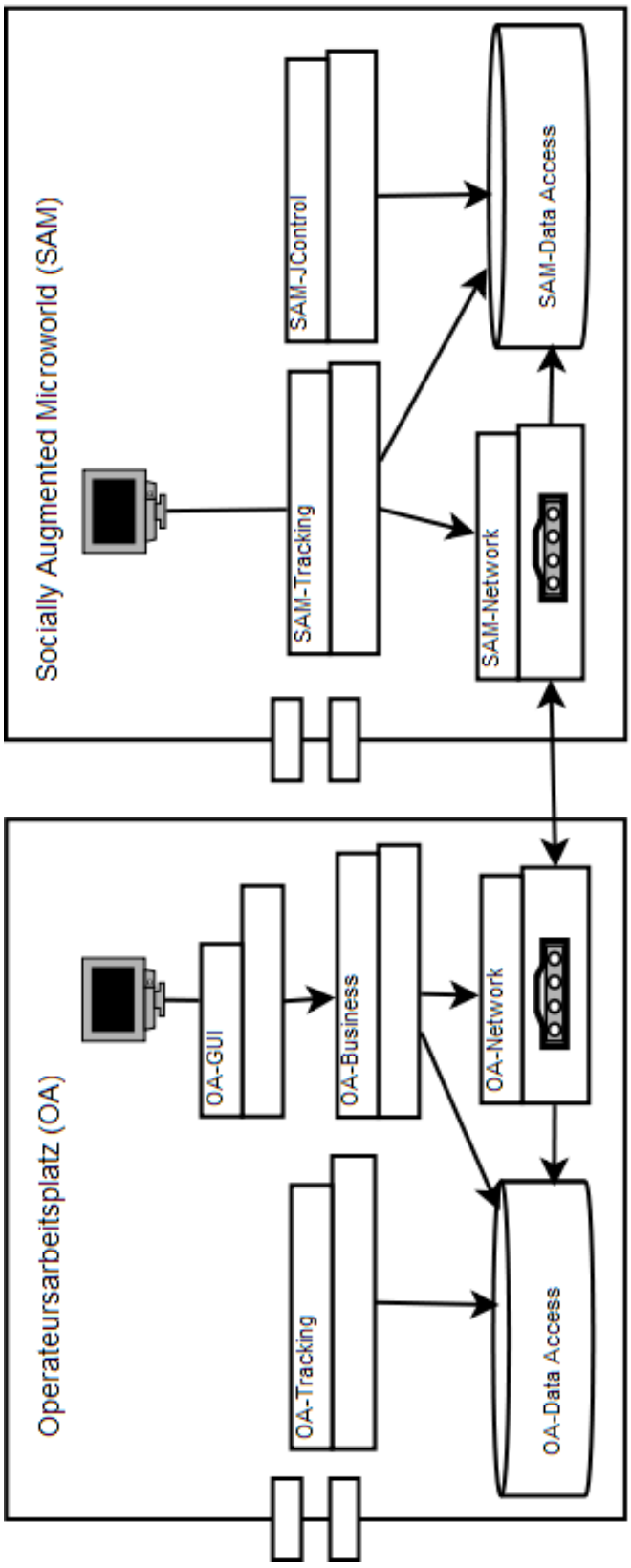

Abbildung 38: Softwarearchitektur von OA

C.2 VERERBUNGS- UND ASSOZIATIONSBEZIEHUNGEN IM OA

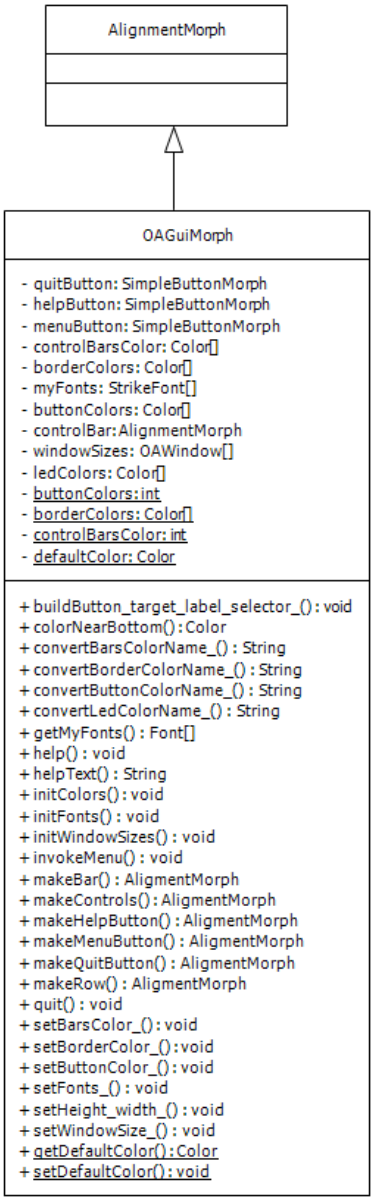

Abbildung 39: Klasse *OAGuiMorph*

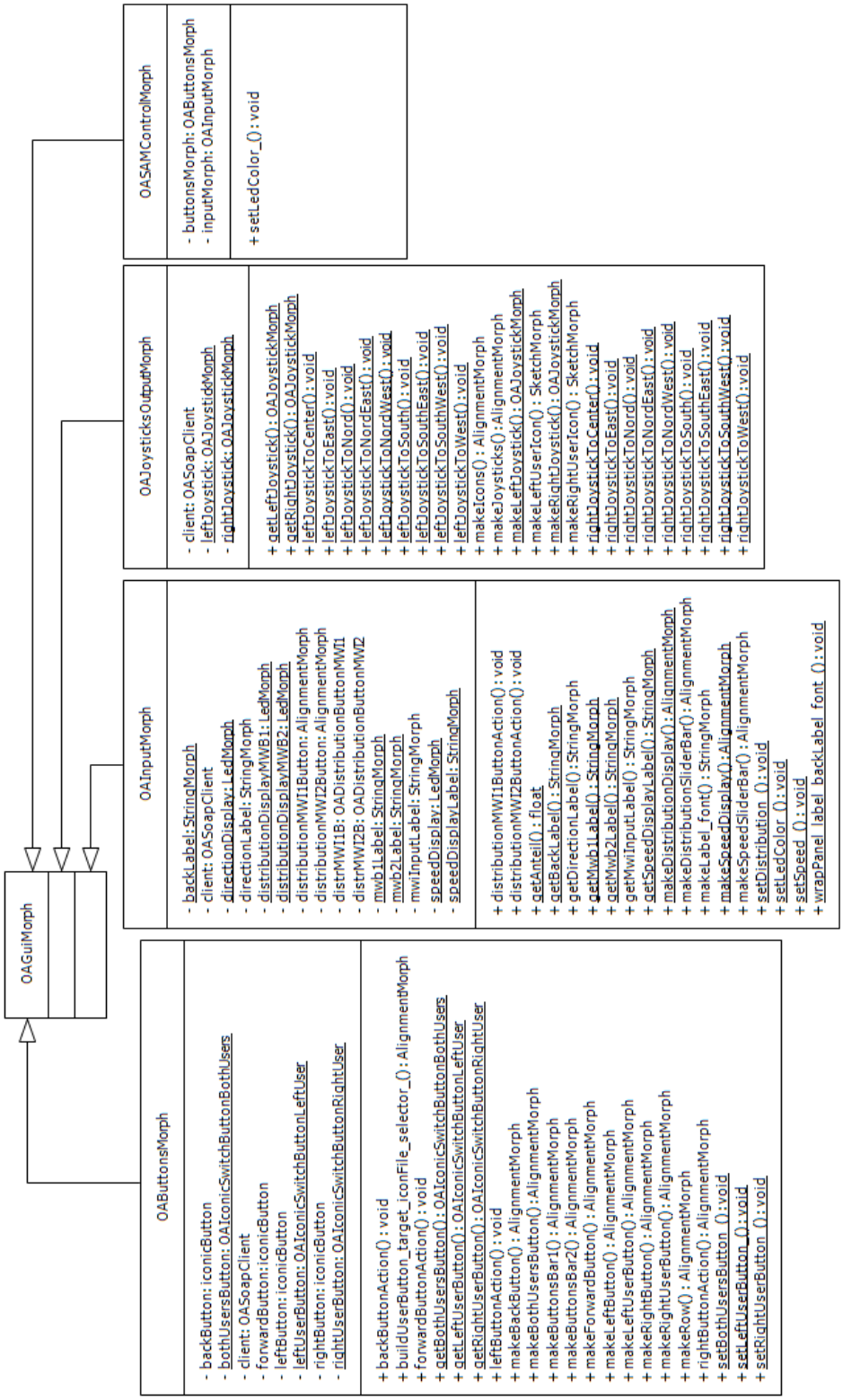

Abbildung 40: Klasse *OAGuiMorph*

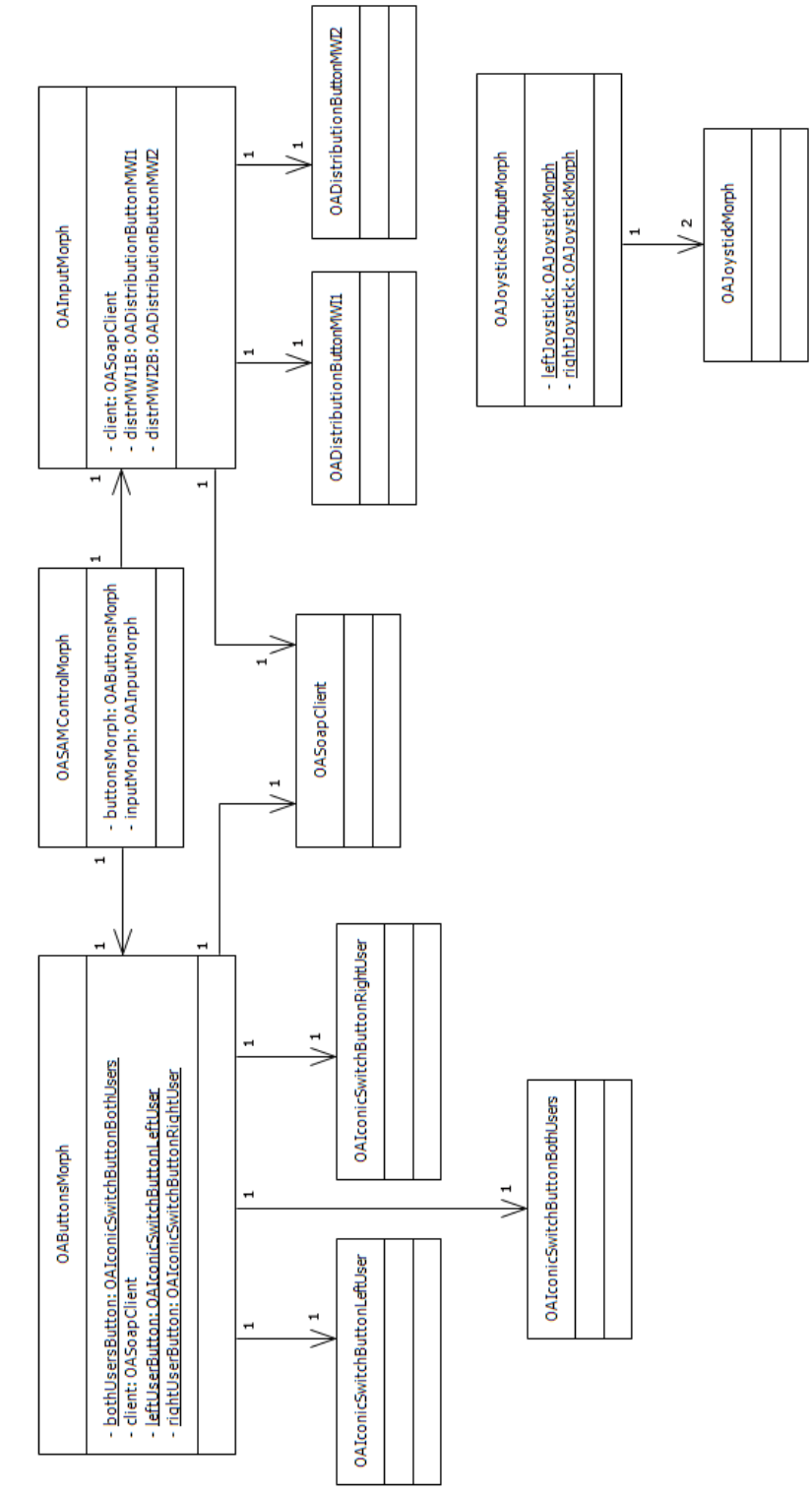

Abbildung 41: Assoziationen

D

KLASSEN UND METHODEN VON OA

D.1 KLASSEN

In der Tabelle 10 werden zunächst alle für den OA entworfenen Klassen beschrieben. In darauf folgenden Abschnitten werden für jede dieser Klassen die von diesen Klassen implementierten Instanz- und Klassenmethoden beschrieben.

Tabelle 10: Klassen des Operateurarbeitsplatzes

Klasse	Beschreibung
`OAButtonsMorph`	GUI zum Senden von auditiven und visuellen Signalen
`OAControl`	Die Klasse stellt Methoden zum Laden sowohl der Konfigurationen und Grafiken als auch zum Starten des Trackings zur Verfüngung
`OADistributionButtonMWI1`	Button zum Erhöhen des Inputanteils von MWB1 (Inputanteil von MWB2 wird entsprechend gesenkt)
`OADistributionButtonMWI2`	Button zum Erhöhen des Inputanteils von MWB2 (Inputanteil von MWB1 wird entsprechend gesenkt)
`OAGuiMorph`	Template für alle entworfenen GUIs
`OAIconicSwitchButtonBothUsers`	Button zum Umschalten des Signalempfängers auf beide MWB

Tabelle 10: Klassen des Operateurarbeitsplatzes

Klasse	Beschreibung
`OAIconicSwitchButtonLeftUser`	Button zum Umschalten des Signalempfängers auf ausschließlich MWB1
`OAIconicSwitchButtonRightUser`	Button zum Umschalten des Signalempfängers auf ausschließlich MWB2
`OAInputMorph`	GUI zum Ändern des Inputanteils beider MWB, zum indirekten Eingreifen in die Steuerung und zum Ändern der maximalen Geschwindigkeit
`OAJoystickMorph`	Morph zum Visualisieren der Joystickbewegungen
`OAJoysticksOutputMorph`	GUI zum Visualisieren der Joystickbewegungen
`OASAMControlMorph`	GUI für die Eingriffe in das System SAM (enthält *OAButtonsMorph* und *OAInputMorph*)
`OASAMData`	Dient als Datencontainer zum Cachen von Daten, die von SAM kommen, aber auch zur Speicherung von initialisierenden Daten
`OASliderMorph`	Schieberegler
`OASoapClient`	Methoden zum Aufrufen von Web Services von SAM
`OASoapServices`	Web Services von OA

Tabelle 10: Klassen des Operateurarbeitsplatzes

Klasse	Beschreibung
`OATrackingFrame`	ein Rahmen zum Hervorheben des Streckenbereiches, welcher für beide MWB sichtbar ist
`OAWindow`	strukturierter Datentyp zum Speichern von Parametern eines Fensters

D.2 METHODEN DER KLASSEN

Die in der Tabelle 10 angegebenen Klassen implementieren Methoden, welche in den folgenden Unterabschnitten beschrieben werden. Die Methode `initialize` wird von jeder Klasse implementiert und wird immer dann aufgerufen, wenn eine Instanz der jeweiligen Klasse gebildet wird. Aus diesem Grund wird diese Methode nicht explizit für jede Klasse beschrieben. Einige Klassen besitzen keine Instanz- oder keine Klassenmethoden.

D.2.1 *OAButtonsMorph*

Tabelle 11: Instanzmethoden der Klasse *OAButtonsMorph*

Methode	Beschreibung
`backButtonAction`	Eventbehandlung des Backbuttons
`buildButton:target:iconFile:selector:`	Erzeugung eines Wrappers für einen Button, um einen Raum um den Button zu erzeugen. Einfügen des Buttons in den Wrapper. Zurückgeben des Wrappers inklusive des Buttons
`buildUserButton:target:iconFile:selector:`	wie `buildButton:target:iconFile:selector:` Farbe des Buttons wird von dieser Methode nicht überschrieben
`down`	Eventbehandlung des Helpbuttons
`forwardButtonAction`	Eventbehandlung des Forwardbuttons
`leftButtonAction`	Eventbehandlung des Leftbuttons

Tabelle 11: Instanzmethoden der Klasse *OAButtonsMorph*

Methode	Beschreibung
`makeBackButton`	Erzeugen und Zurückgeben des Backbuttons
`makeBothUsersButton`	Erzeugen und Zurückgeben des Bothusersbuttons, welcher den Sendeempfang von Signalen auf beide MWB setzt
`makeButtonsBar1`	Erzeugen der Buttonleiste mit den Left-, Right-, Forward- und Backbuttons
`makeButtonsBar2`	Erzeugen der Buttonleiste mit den Buttons zum Umschalten des Signalempfängers
`makeForwardButton`	Erzeugen und Zurückgeben des Forwardbuttons
`makeLeftButton`	Erzeugen und Zurückgeben des Leftbuttons
`makeLeftUserButton`	Erzeugen und Zurückgeben des Leftusersbuttons, welcher den Sendeempfang von Signalen auf MWB1 setzt
`makeRightButton`	Erzeugen und Zurückgeben des Rightbuttons

Tabelle 11: Instanzmethoden der Klasse *OAButtonsMorph*

Methode	Beschreibung
`makeRightUserButton`	Erzeugen und Zurückgeben des Leftusersbuttons, welcher den Sendeempfang von Signalen auf MWB2 setzt
`makeRow`	Private Methode zum Erzeugen einer horizontalen Leiste
`rightButtonAction`	Eventbehandlung des Rightbuttons

Tabelle 12: Klassenmethoden der Klasse *OAButtonsMorph*

Methode	Beschreibung
`getBothUsersButton`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `bothUsersButton`, welche das entsprechende Buttonobjekt enthält
`getLeftUserButton`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `leftUserButton`, welche das entsprechende Buttonobjekt enthält
`getRightUserButton`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `rightUserButton`, welche das entsprechende Buttonobjekt enthält
`setBothUsersButton`	Setter-Methode zum Definieren des Wertes der Klassenvariable `bothUsersButton`, welche das entsprechende Buttonobjekt enthält

Tabelle 12: Klassenmethoden der Klasse *OAButtonsMorph*

Methode	Beschreibung
`setLeftUserButton`	Setter-Methode zum Definieren des Wertes der Klassenvariable `leftUserButton`, welche das entsprechende Buttonobjekt enthält
`setRightUserButton`	Setter-Methode zum Definieren des Wertes der Klassenvariable `rightUserButton`, welche das entsprechende Buttonobjekt enthält

D.2.2 *OAControl*

Tabelle 13: Klassenmethoden der Klasse *OAControl*

Methode	Beschreibung
`loadConfigsAndGraphics`	Laden der Konfigurationen und Grafiken in den Arbeitsspeicher
`quit`	Abbrechen des Trackings
`setInputDevice:`	Setzen des Eingabegeräts (Joystick oder Tastatur)
`startTracking`	Starten des Trackings in OA

D.2.3 *OADistributionButtonMWI1 und OADistributionButtonMWI2*

Tabelle 14: Klassenmethoden der Klassen *OADistributionButtonMWI1* und *OADistributionButtonMWI2*

Methode	Beschreibung
`getState`	Zurückgeben des aktuellen Zustandes von *OADistributionButtonMWI*[1|2] (Zustandsraum $Z = \{$"'*supersmall*"', "'*small*"', "'*normal*"', "'*big*"', "'*superbig*"'$\}$)
`setState:`	Ändern des aktuellen Zustandes von *OADistributionButtonMWI*[1|2]

D.2.4 *OAGuiMorph*

Tabelle 15: Instanzmethoden der Klasse *OAGuiMorph*

Methode	Beschreibung
`buildButton:target:label:selector:`	Erzeugung eines Wrappers für einen Button, um einen Raum um den Button zu erzeugen. Einfügen des Buttons in den Wrapper. Zurückgeben des Wrappers inklusive dea Buttons
`defaultColor`	Getter-Methode zum Zurückgeben einer in dieser Methode definierten RGB-Farbe
`convertBarsColorName:`	Private Methode zum Ändern einer Leistenfarbenbezeichnung im RGB-Format zu einem beliebigen String

Tabelle 15: Instanzmethoden der Klasse *OAGuiMorph*

Methode	Beschreibung
`convertBorderColorName:`	Private Methode zum Ändern einer Randfarbenbezeichnung zu einem beliebigen String
`convertButtonColorName:`	Private Methode zum Ändern einer Buttonfarbenbezeichnung zu einem beliebigen String
`convertLedColorName:`	Private Methode zum Ändern einer Farbenbezeichnung von LED-Anzeige zu einem beliebigen String
`getMyFonts`	Getter-Methode zum Zurückgeben des Wertes der Instanzvariable `myFonts`
`help`	Eventbehandlung des Helpbuttons
`helpText`	Zurückgeben des Textes, welcher vom Hilfefenster benutzt wird
`initColors`	Initialisieren der Farben für den Rand von GUIs, Leisten, Buttons und LEDs in den entsprechenden Instanzvariablen
`initFonts`	Initialisieren der Schriftgrößen in der entsprechenden Instanzvariable
`initWindowSizes`	Initialisieren von Fenstergrößen in der entsprechenden Instanzvariable

Tabelle 15: Instanzmethoden der Klasse *OAGuiMorph*

Methode	Beschreibung
`invokeMenu`	Erzeugen aller Menüpunkte fürs Hauptmenü von `OAGuiMorph`
`makeBar`	Erzeugen und Zurückgeben einer horizontalen Leiste
`makeControls`	Erzeugen und Zurückgeben der Menüleiste
`makeHelpButton`	Erzeugen und Zurückgeben des Hilfebuttons
`makeMenuButton`	Erzeugen und Zurückgeben des Menübuttons
`makeQuitButton`	Erzeugen und Zurückgeben des Beendenbuttons
`makeRow`	Private Methode zum Erzeugen einer horizontalen Leiste
`quit`	Eventbehandlung des Buttons `quit`
`setBarsColor:`	Definieren der Farbe von `OAGuiMorph`
`setBorderColor:`	Definieren der Randfarbe von `OAGuiMorph`
`setButtonColor:`	Definieren der Farbe aller Buttons von `OAGuiMorph`
`setFonts:`	Definieren der Schriftgrößen von `OAGuiMorph`
`setHeight:width:`	Private Methode zum Definieren Fenstergröße von *OAGuiMorph*

Tabelle 15: Instanzmethoden der Klasse *OAGuiMorph*

Methode	Beschreibung
`setWindowSize:`	Definieren Fenstergröße von *OAGuiMorph*

D.2.5 *OAIconicSwitchButtonBothUsers, OAIconicSwitchButtonLeftUser und OAIconicSwitchButtonRightUser*

Tabelle 16: Instanzmethoden der Klassen *OAIconicSwitchButtonBothUsers OAIconicSwitchButtonLeftUser* und *OAIconicSwitchButtonRightUser*

Methode	Beschreibung
`borderInset`	Setzen oder Ändern des Randstils des Buttons im gedrückten Zustand
`borderNormal`	Setzen oder Ändern des Randstyles des Buttons im neutralen Zustand
`borderRaised`	Setzen oder Ändern des Randstyles des Buttons im losgelassenen Zustand
`borderThick`	Setzen oder Ändern des Randstyles des Buttons beim Betreten der Maus von Dimensionen des Buttons
`doButtonAction`	Eventbehandlung des Buttons
`labelFromString:`	Definieren eines Buttonlabels als Zeichenkette
`labelGraphic:`	Definieren eines Buttonlabels als Grafik
`mouseEnter:`	Eventbehandlung des Ereignisses, wenn der Mauszeiger in die Grenzen des Buttons eintritt

Tabelle 16: Instanzmethoden der Klassen *OAIconicSwitchButtonBothUsers OAIconicSwitchButtonLeftUser* und *OAIconicSwitchButtonRightUser*

Methode	Beschreibung
`mouseLeave:`	Eventbehandlung des Ereignisses, wenn der Mauszeiger den Bereich des Buttons verlässt
`mouseUp:`	Eventbehandlung des Ereignisses, wenn die Maustaste über dem Button losgelassen wird
`setDefaultLabel`	Setzen des Buttonlabels auf die Standardgrafik
`setSwitchState:`	Aktionen beim Wechsel des Zustandes durch Betätigen der Userbuttons

Tabelle 17: Klassenmethoden der Klassen *OAIconicSwitchButtonBothUsers*, *OAIconicSwitchButtonLeftUser* und *OAIconicSwitchButtonRightUser*

Methode	Beschreibung
`offColor`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `offColor`, welche die Buttonfarbe im ausgeschalteten Zustand enthält
`offColor:`	Setter-Methode zum Zuweisen des Wertes der Klassenvariable `offColor`, welche die Buttonfarbe im ausgeschalteten Zustand enthält
`onColor`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `offColor`, welche die Buttonfarbe im eingeschalteten Zustand enthält

Tabelle 17: Klassenmethoden der Klassen *OAIconicSwitchButtonBothUsers*, *OAIconicSwitchButtonLeftUser* und *OAIconicSwitchButtonRightUser*

Methode	Beschreibung
`onColor:`	Setter-Methode zum Zuweisen des Wertes der Klassenvariable `offColor`, welche die Buttonfarbe im eingeschalteten Zustand enthält

D.2.6 *OAInputMorph*

Tabelle 18: Instanzmethoden der Klasse *OAInputMorph*

Methode	Beschreibung
`invokeMenu`	Erzeugen aller Menüpunkte für das Hauptmenü von *OAInputMorph*
`makeDistributionSliderBar`	Erzeugen der Leiste mit dem Schieberegler zum Ändern des Inputanteils von MWB
`makeInputBar`	Erzeugen der Leiste mit dem Schieberegler zum Ändern der maximalen Geschwindigkeit und Erzeugen der Leiste mit dem Schieberegler zum Eingreifen in die Steuerung
`setBarsColor:`	Ändern der Farbe aller Untermorphe von *OAInputMorph*

Tabelle 19: Klassenmethoden der Klasse *OAInputMorph*

Methode	Beschreibung
`defaultColor`	Getter-Methode zum Zurückgeben einer in dieser Methode definierten RGB-Farbe
`getAnteil`	Zurückgeben des LED-Displaywertes, um es zum Initialisieren der Schiebereglerpositionen zu verwenden
`makeBar`	Erzeugen und Zurückgeben einer horizontalen Leiste
`makeBarWithBorderWidth:`	Erzeugen und Zurückgeben einer horizontalen Leiste mit definierter Randgröße
`makeDistributionDisplay`	Erzeugen und Zurückgeben der Leiste mit dem LED zur Ausgabe des Inputanteils von MWB
`makeSpeedDisplay`	Erzeugen und Zurückgeben der Leiste mit dem LED zur Ausgabe der maximalen Geschwindigkeit
`setBarsColor:`	Definieren der Farbe aller Leisten von *OAInputMorph*
`setDistribution:`	Definieren des Displaywertes der LEDs zum Anzeigen der Inputanteile
`setLedColor:`	Definieren der Farbe aller LED-Displays von *OAInputMorph*
`setSpeed:`	Definieren des Displaywertes eines LEDs zum Anzeigen der maximalen Geschwindigkeit

Tabelle 19: Klassenmethoden der Klasse *OAInputMorph*

Methode	Beschreibung
`wrapPanel:label:backLabel:`	Private Methode zum Erzeugen eines Wrappers für die Beschriftung eines LED-Displays

D.2.7 *OAJoystickMorph*

Tabelle 20: Instanzmethoden der Klasse *OAJoystickMorph*

Methode	Beschreibung
`getHandle`	Getter-Methode zum Zurückgeben der Instanzvariable handleMorph, welche das Joystickobjekt enthält

Tabelle 21: Klassenmethoden der Klasse *OAJoystickMorph*

Methode	Beschreibung
`center`	Getter-Methode zum Zurückgeben der relativen Koordinaten vom Zentrum des quadratischen Morphes, in welchem die Joystickbewegungen visualisiert werden

D.2.8 *OAJoysticksOutputMorph*

Tabelle 22: Instanzmethoden der Klasse *OAJoysticksOutputMorph*

Methode	Beschreibung
`makeJoysticks`	Erzeugen eines Wrappers und Einfügen der erzeugten Joystickmorphe in selbigen
`setBarsColor:`	Definieren der Farbe von *OAJoysticksOutputMorph*

Tabelle 23: Klassenmethoden der Klasse *OAJoysticksOutputMorph*

Methode	Beschreibung
`getLeftJoystick`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `leftJoystick`, welche das entsprechende Joystickobjekt enthält
`getRightJoystick`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `rightJoystick`, welche das entsprechende Joystickobjekt enthält
[`left`|`right`]`JoystickToCenter`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in das **Zentrum** von *OAJoystickmorph*
[`left`|`right`]`JoystickToEast`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in die Mitte des **rechten** Randes von *OAJoystickmorph*

Tabelle 23: Klassenmethoden der Klasse *OAJoysticksOutputMorph*

Methode	Beschreibung
`[left|right]JoystickToNorth`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in die Mitte des **oberen** Randes von *OAJoystickmorph*
`[left|right]JoystickToNorthEast`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in die **rechte obere** Ecke von *OAJoystickmorph*
`[left|right]JoystickToNorthWest`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in die **linke obere** Ecke von *OAJoystickmorph*
`[left|right]JoystickToSouth`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in die Mitte des **unteren** Randes von *OAJoystickmorph*
`[left|right]JoystickToSouthEast`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in die **rechte untere** Ecke von *OAJoystickmorph*
`[left|right]JoystickToSouthWest`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in die **linke untere** Ecke von *OAJoystickmorph*
`[left|right]JoystickToWest`	Positionieren der Visualisierung des Joysticks von MWB[1|2] in die Mitte des **linken** Randes von *OAJoystickmorph*
`makeLeftJoystick`	Erzeugen und Zurückgeben des Joystickmorphes für MWB1
`makeRightJoystick`	Erzeugen und Zurückgeben des Joystickmorphes für MWB2

Tabelle 23: Klassenmethoden der Klasse *OAJoysticksOutputMorph*

Methode	Beschreibung

D.2.9 *OASAMControlMorph*

Tabelle 24: Instanzmethoden der Klasse *OASAMControlMorph*

Methode	Beschreibung
`invokeMenu`	Erzeugen aller Menüpunkte für das Hauptmenü von *OASAMControlMorph*
`setBarsColor:`	Definieren der Farbe von *OASAMControlMorph*
`setFonts:`	Definieren der Schriftgrößen von *OASAMControlMorph*
`setLedColor:`	Definieren der Farbe aller LED-Displays von *OASAMControlMorph*

D.2.10 *OASAMData*

Tabelle 25: Klassenmethoden der Klasse *OASAMData*

Methode	Beschreibung
`getCarCoordinates`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `carCoordinates`, welche die Koordinaten des Fahrobjektes enthält
`getCarExtent`	Gibt die Dimensionen des Fahrobjektes (Durchschnitt) zurück

Tabelle 25: Klassenmethoden der Klasse *OASAMData*

Methode	Beschreibung
`getSpeed`	Getter-Methode zum Zurückgeben des Wertes der Klassenvariable `speed`, welche die Geschwindigkeit des Fahrobjektes enthält
`getSpeedColor`	Implementierung der Abbildung der Geschwindigkeit auf die Farben
`setCarCoordinates:`	Setter-Methode zum Setzen des Wertes der Klassenvariable `carCoordinates`
`setSpeed:`	Setter-Methode zum Setzen des Wertes der Klassenvariable `speed`

D.2.11 *OASliderMorph*

Tabelle 26: Instanzmethoden der Klasse *OASliderMorph*

Methode	Beschreibung
`setSliderThickness:`	Setter-Methode zum Definieren des Wertes der Instanzvariable `slTh`, welche als Wert die Breite des beweglichen Teiles des Schiebereglers enthält
`sliderThickness`	Getter-Methode zum Zurückgeben der Instanzvariable `slTh`

D.2.12 *OASoapClient*

Tabelle 27: Instanzmethoden der Klasse *OASoapClient*

Methode	Beschreibung
`callButtonService:`	Aufrufen eines SOAP-Services
`hostAddress:`	Definieren der IP-Adresse der Maschine, welche SOAP Services anbietet

D.2.13 *OATrackingFrame*

Tabelle 28: Klassenmethoden der Klasse *OATrackingFrame*

Methode	Beschreibung
`deleteFrame`	Entfernen des Rahmens

D.2.14 *OAWindow*

Tabelle 29: Instanzmethoden der Klasse *OAWindow*

Methode	Beschreibung
`get[Height|Width]`	Getter-Methode zum Zurückgeben des Wertes der Instanzvariable `[height|width]`, welche die [Höhe|Breite] eines Morphobjektes enthält
`set[Height:|Width:]`	Setter-Methode zum Definieren der Instanzvariable `[height|width]`, welche die [Höhe|Breite] eines Morphobjektes enthält

E

QUELLCODE

E.1 IMPLEMENTIERUNG DES SCHWEIFES

Die Darstellung des Schweifes wird in der Methode *OA.ATEOAblaufsteuerung»step* (Inputmethode 'Tastatur') oder *OA.ATEOrealJoysticksStepping»step* (Inputmethode 'Joystick') implementiert. Im Listing E.1 wird die Methode *OA.ATEOAblaufsteuerung»step* gezeigt. Der Schweif wird in den Zeilen 12 bis 49 implementiert. Dabei werden zunächst zwei Kreise mit Hilfe der Klasse *EllipseMorph* erzeugt (Zeilen 29, 35) und als nächstes werden diese Kreise in die momentan befahrenen Spielbretter als Submorphe hinzugefügt (Zeilen 45, 48).

```
1  step
2
3  | circle1 circle2 bild1 bild2 circlePosX circlePosY |
4
5  "schritt - speed. The value of schritt is sent by SAM
6   punkt - position of the Car.
7   The value of punkt is sent by SAM"
8
9  self step:5 punkt: 10@10.
10
11 "************************************"
12 stepsAmount := stepsAmount + 1.
13
14 "after 30 steps remove all the circles of the tail"
15 (stepsAmount \\ 30 = 0)
16 ifTrue:
17 [
18  ParID bild1 removeAllMorphs.
19  ParID bild2 removeAllMorphs.
20 ].
21
22 "every 1-st step add a circle"
23 (stepsAmount \\ 1 = 0)
24 ifTrue:
25 [
26 circlePosX := (CarID aktpos) x - (15 / 2).
27 circlePosY := CarID aktpos y.
28
29 circle1 := EllipseMorph new.
30 circle1 extent: 15@15;
31       color: Color green;
32       borderWidth:0;
33       position: circlePosX @ circlePosY.
34
35 circle2 := EllipseMorph new.
36 circle2 extent: 15@15;
37       color: Color green;
38       borderWidth:0;
```

```
39      position: circlePosX @ circlePosY.
40
41
42
43
44  bild1 := ParID bild1.
45  bild1 addMorph: circle1.
46
47  bild2 := ParID bild2.
48  bild2 addMorph: circle2.
49  ].
50  "***********************************"
51
52  "CarID startpos: (CarID aktpos x)@(CarID aktpos y)."
53
54  CarID aktposkorr.
55  "Auto vor Verlassen des Bildschirmausschnitts bewahren"
56
57  "Abfragen ob Ende der Strecke erreicht"
58  "(und am Ziel vorbeigefahren)"
59
60  (CarID sensor3 = (Color r: 0.387 g: 1.0 b: 1.0))
61    ifTrue: [
62      VuID fertig.
63      ].
```

Listing E.1: Die Implementierung des Schweifes in der Methode *step*

E.2 SIMULATION ZUM TESTEN DES SCHWEIFES

Es wurde eine Simulation des Trackings entwickelt, um den Schweif ohne eine vorhandene Vernetzung mit SAM zu testen. Die Simulation generiert Geschwindigkeits- und Positionsdaten des Fahrobjektes, welche später mit Hilfe einer Vernetzung von SAM geliefert werden sollen.

Die im Listing E.2 gezeigten Zeilen definieren eine Simulation des Trackings, indem sowohl die Geschwindigkeit als auch die Position des Fahrobjektes mit Hilfe der Sinusfunktion verändert werden. Diese Zeilen müssen in *OA.ATEOAblaufsteuerung»step* (Inputmethode 'Tastatur') oder in *OA.ATEOrealJoysticksStepping»step* (Inputmethode 'Joystick') eingefügt werden, um eine Simulation durchführen zu können.

```
1  "for changing of speed between 3 and 27 (sinus, average
       by 15)"
2  OASAMData setSpeed: 15 + (12 * ((stepsAmount / 20) sin)).
3
4  OASAMData setCarCoordinates: 400@400.
5
6  currentSpeed := OASAMData getSpeed.
7  currentSpeedColor := OASAMData getSpeedColor.
8  currentCarCoordinates := OASAMData getCarCoordinates.
9
10 "to move the car to the left and to the right all the
       time (sinus)"
```

```
11 currentX := (currentCarCoordinates x) + (110 * ((
       stepsAmount / 20) sin)).
12 currentY := currentCarCoordinates y.
13 currentCarCoordinates := currentX@currentY.
14
15 carExtent := OASAMData getCarExtent.
16
17 "schritt - speed. The value of schritt is sent by SAM
18 punkt - position of the Car. The value of punkt is sent
       by SAM"
19 Ctrl step:currentSpeed punkt: currentCarCoordinates.
```

Listing E.2: Simulation zum Testen des Schweifes

E.3 SKALIEREN DER SPIELBRETTER

Die Anweisung *scaledToSize:* in der Methode *OA.ATEOPar»bilderLaden:* (Zeile 13) kann die Spielbretter skalieren.

```
1  bilderLaden: aFile
2    "...."
3    "...."
4    "...."
5
6    seqColl
7       do: [:each |
8       dindex := seqColl indexOf: each.
9       imageArray
10      at: dindex
11      put: (ImageMorph new image:
12      "(Form fromFileNamed: each)" "replaced with next line
            "
13      (Form fromFileNamed: each) scaledToSize: 600@1500 )].
14
15   "...."
16   "...."
```

Listing E.3: Skalieren der Spielbretter

LITERATURVERZEICHNIS

DIN ISO 9126. Informationstechnik - Beurteilen von Softwareprodukten, Qualitätsmerkmale und Leitfaden zu deren Verwendung. Berlin: Beuth-Verlag, 1991. (Zitiert auf Seiten ix, 19, and 20.)

EN ISO 9241-11. Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten, Teil 11: Anforderungen an die Gebrauchstauglichkeit - Leitsätze. Berlin: Beuth-Verlag, 1998. (Zitiert auf Seite 19.)

Helmut Balzert. *Lehrbuch der Software-Technik. Software-Entwicklung*. Spektrum Akademischer Verlag, 2000. ISBN 3-8274-0480-0. (Zitiert auf Seiten 13, 18, 19, 21, 22, 37, and 41.)

Simon Banbury und Sébastien Tremblay. *A Cognitive Approach to Situation Awareness: Theory and Application*. Ashgate Publishing, Ltd., 2004. ISBN 9780754641988. (Zitiert auf Seite 14.)

Lon Barfield. *The User Interface: Concepts and Design*. Bosko Books, 2004. ISBN 978-0954723903. (Zitiert auf Seite 10.)

H. H. Bell und D. R. Lyon. Using observer ratings to assess situation awareness. In M. R. Endsley und D. J. Garland, editors, *Situation awareness analysis and measurement*. Lawrence Erlbaum, Mahwah, NJ, 2000. (Zitiert auf Seite 31.)

David Booth und Hugo Haas. Web services architecture. http://www.w3.org/TR/ws-arch/#service_oriented_architecture; 17.09.08, 2004. (Zitiert auf Seite 9.)

Alf Ove Braseth, Robin Welch, und Øystein Veland. A building block for information rich displays. Technical report, IFE Halden, November 2003. (Zitiert auf Seiten 25 and 27.)

Catherine M. Burns und John R. Hajdukiewicz. *Ecological interface design*. CRC Press, 2004. ISBN 0-415-28374-4. (Zitiert auf Seite 25.)

Francis T. Durso und Arathi Sethumadhavan. Situation awareness: Understanding dynamic environments. *Texas Tech University, Lubbock, Texas*, 2008. (Zitiert auf Seite 25.)

M. R. Endsley. Design and evaluation for situation awareness enhancement. pages pp.97–101. Proceedings of the Human Factors Society 30th Annual Meeting, Santa Monica, CA: Human Factors Society, 1988. (Zitiert auf Seite 14.)

M. R. Endsley, Betty Bolté, und Debra G. Jones. *Designing for Situation Awareness*. Taylor & Francis, 2003. ISBN 0-748-40966-1. (Zitiert auf Seiten 1, 9, 29, 30, 31, and 37.)

Mica R. Endsley. Situation awareness and human error: Designing to support human performance. Technical report, SA Technologies, Inc., 1999. (Zitiert auf Seite 26.)

Norman E. Fenton und Shari L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach, Revised*. Course Technology, 1998. (Zitiert auf Seite 32.)

N. Hamacher, K.-F. Kraiss, und J. Marrenbach. Einsatz formaler methoden zur evaluierung der gebrauchsfähigkeit interaktiver geräte. *it + ti Informationstechnik und Technische Informatik*, 44(1):49–55, 2002. (Zitiert auf Seite 28.)

Nico Hamacher und Karl-Friedrich Kraiss. Expertensystem zur kritierienorientierten bewertung der gebrauchsfähigkeit von dialogsystemen. In *VDE-Kongress "'Innovationen für Menschen"'*, pages 193–198. VDE Verlag GmbH, 2004. (Zitiert auf Seiten ix, 28, 29, 32, and 69.)

Nicolas Alexander Hamacher. Automatische kriterienorientierte bewertung der gebrauchstauglichkeit interaktiver systeme. Dissertation, Rheinisch-Westfälische Technische Hochschule Aachen, 2006. (Zitiert auf Seite 28.)

Sung H. Han, Huichul Yang, und Dong-Gwan Im. Designing a human-computer interface for a process control room: A case study of a steel manufacturing company. *International Journal of Industrial Ergonomics. 37*, 2007. (Zitiert auf Seite 27.)

Michael Herczeg. *Software-Ergonomie. Grundlagen der Mensch-Computer-Kommunikation*. Addison-Wesley, 1994. ISBN 3-89319-615-3. (Zitiert auf Seite 3.)

Munira Jessa und Burns Catherine M. Visual sensitivities of dynamic graphical displays. *Int. J. Human-Computer Studies 65*, 2007. (Zitiert auf Seite 24.)

Bernd Kahlbrandt. *Software Engineering*. Springer-Verlag GmbH, Berlin. ISBN 354063309X. (Zitiert auf Seite 19.)

Malik M. A. Khalfan und Chimay J. Anumba. Implementation of concurrent engineering in construction - readiness assessment. Technical report, Department of Civil & Building Engineering, Loughborough University, 1999. (Zitiert auf Seite 39.)

Jochen Ludewig und Horst Lichter. *Software Engineering*. dpunkt.verlag GmbH, Heidelberg. ISBN 3898642682. (Zitiert auf Seite 39.)

John Maloney. An introduction to morphic: The squeak user interface framework. In *Squeak: Open Personal Computing and Multimedia*, pages 39–67. Prentice Hall, 2002. (Zitiert auf Seite 70.)

Faouzi Moussa und Meriem Riahi. Use of formalized guidelines for semi-automated generation of gui: the ergo-conceptor+ tool. In Jean Vanderdonckt und Christelle Farenc, editors, *Tools for Working with Guidelines*. Springer, 2000. (Zitiert auf Seite 28.)

J.M O'Hara, W.S. Brown, P.M. Lewis, und J.J. Persensky. Human-system interface design review guidelines (nureg-0700, rev. 2). Technical report, Energy Sciences and Technology Department Brookhaven National Laboratory, May 2002. (Zitiert auf Seiten 23, 24, 25, 26, 27, and 28.)

C.R. Pedersen und M. Lind. Conceptual design of industrial process displays. *Ergonomics, Volume 42, Number 11*, 1999. (Zitiert auf Seiten 26 and 27.)

E. Piso. Task analysis for process-control tasks: the method of annett et al. applied. *Journal of Occupational Psychology.54*, 1981. (Zitiert auf Seite 12.)

Jeannie L. Pridmore. Designing for the improvement of operator situation awareness in automation systems. Master's thesis, Auburn University, 2007. (Zitiert auf Seite 27.)

M. Rauterberg, S. Schluep, und M. Fjeld. Modelling of cognitive complexity with petry nets: an action theoretical approach. In R. Trappl, editor, *Cybernetics and Systems*. Wien: Austrian Society for Cybernetic Studies, 1998. (Zitiert auf Seite 32.)

M. Rauterberg, O. Strohm, und C. Kirsch. Benefits of user-oriented software development based on an iterative cyclic process model for simultaneous engineering. *International Journal of Industrial Ergonomics*, 16:391–410, 1995. (Zitiert auf Seite 39.)

John S. Rhodes, Gregory E. Benoit, und Payne David G. Factors affecting memory for dynamically changing system parameters: implications for interface design. page 284. Proceedings of the IEA 2000/HFES Congress, ProQuest Psychology Journals, 2000. (Zitiert auf Seite 26.)

Andrew Sears. Aide: A step toward metric-based interface development tools. In *ACM Symposium on User Interface Software and Technology*, pages 101–110, 1995. (Zitiert auf Seite 28.)

Neville A. Stanton. Hierarchical task analysis: Developments, applications, and extensions. Technical report, BITlab, Human

Factors Integration Defence Technology Centre, School of Engineering and Design, Brunel University, Uxbridge, June 2005. (Zitiert auf Seite 10.)

M. Vidulich. Testing the sensitivity of situation awareness metrics in interface evaluations. In M. R. Endsley und D. J. Garland, editors, *Situation awareness analysis and measurement*. Lawrence Erlbaum, Mahwah, NJ, 2000. (Zitiert auf Seite 30.)

Y. Zhang und et al. Effects of integrated graphical displays on situation awareness in anaesthesiology. *Cognition, Technology and Work*, 2002. (Zitiert auf Seiten 27 and 37.)

ERKLÄRUNG

Ich versichere hiermit, dass ich meine Diplomarbeit mit dem Thema

> *Fenster zum Prozess: ein Operateursarbeitsplatz zur Überwachung und Kontrolle von kooperativem Tracking*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Diplomarbeit zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in dreifacher Ausfertigung und gebunden im Prüfungsamt der Humboldt Universität zu Berlin abzugeben oder spätestens mit dem Poststempel des Tages, an dem die Frist abläuft, zu senden habe.

Berlin, den 21. 1. 2009

Hermann Schwarz