Fenster zum Prozess: Weiterentwicklung eines Operateursarbeitsplatzes im Projekt Arbeitsteilung Entwickler Operateur (ATEO)

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von:	Christian Leonhard
geboren am:	05.11.1977
in:	Hennigsdorf
Gutachter:	Prof. Klaus Bothe
	Prof. Jens Nachtwei
eingereicht am:	.....

Christian Leonhard: *FENSTER ZUM PROZESS: WEITERENTWICKLUNG EINES OPERATEURSARBEITSPLATZES IM PROJEKT ARBEITSTEILUNG ENTWICKLER OPERATEUR (ATEO),* Diplomarbeit, © Januar 2013

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

AKRONYME

ATEO Arbeitsteilung Entwickler Operateur

prometei Prospektive Gestaltung von Mensch-Technik-Interaktion

SAM Socially Augmented Microworld

OA Operateursarbeitsplatz

MA Mentale Anstrengung

MWB Mikroweltbewohner

GUI Graphical User Interface

TCP/IP Transmission Control Protocol / Internet Protocol

UDP User Datagram Protocol

HTA Hierarchical Task Analysis

OOA Objektorientierte Analyse

IEA International Ergonomics Association

1 EINLEITUNG

1.1 EINORDNUNG UND MOTIVATION

Zur Prozessüberwachung und -führung eines dynamischen und komplexen Systems benötigt ein Operateur Schnittstellen zum technischen System. Beispiele für Operateure sind Fluglotsen. Beispiele für Systeme sind industrielle Fertigungsanlagen oder Transportsysteme.

Die Gestaltung der Schnittstellen muss derart erfolgen, dass der Operateur in der Lage ist, Störfälle zu erfassen und zu berichtigen. Durch technische Möglichkeiten können große Informationsmengen aus Systemen bereitgestellt werden. Allerdings würde es den Operateur überfordern, wenn ihm die gesamten Informationen zur Verfügung stünden. Jedoch wäre ein Mangel von Informationen ebenso problematisch wie eine Überflutung damit [14]. Erschwerend kommt hinzu, dass ein Störfall gegebenenfalls mehr Informationen erfordert als der Nomalzustand eines Systems. Die zentrale Herausforderung ist daher die angemessene Dosierung der Informationsmenge.

Im Rahmen des Projektes Arbeitsteilung Entwickler Operateur (ATEO) [23] werden zwei Personengruppen, Operateure und Entwickler, betrachtet. Die Anforderung an beide Gruppen ist die optimale Überwachung und Führung eines Systems. Operateure und Entwickler setzen weitgehend unterschiedliche Fähigkeiten ein. Trotzdem kommt es z.B. beim Antizipieren und Diagnostizieren von Situationen zu Überschneidungen. Die Störungsdiagnose ist für den Operateur von besonderer Bedeutung. Das ist das Erkennen von Zuständen des zu überwachenden Systems, die von der Norm abweichen. Für die Antizipation steht dem Operateur im Vergleich zum Entwickler erheblich weniger Zeit zur Verfügung. Für den Entwickler steht die Störungsantizipation im Mittelpunkt. Dazu muss er mit dem ihm zur Verfügung stehenden Wissen künftiges Systemverhalten abschätzen.

Welche Ressourcen Operateuren und Entwicklern zur Verfügung stehen müssen, um die Anforderung bestmöglich zu erfüllen, ist die zentrale Fragestellung. Diese sind für beide Gruppen verschieden. Die Ressource für die Operateure ist die Schnittstelle zum System. Im Projekt ATEO ist das der Operateursarbeitsplatz, der Gegenstand dieser Diplomarbeit. In der finalen Version des Operateursarbeitsplatzes wird die Ressource über die Quantität der dargebotenen Informationen zu variieren sein. Die Entwickler konzipieren und setzen Automatiken um. Deren Leistungen sind

die Grundlage des Vergleichs beider Gruppen. Als Ressource stehen den Entwicklern Informationen über das System zur Verfügung. Diese werden beim Entwurf der Automatiken verwendet. Variiert wird durch die Menge an verfügbaren Informationen.

Der Entwickler verfügt über Wissen des zu implementierenden Systems und kennt zudem mögliche Vorsysteme. Im Gegensatz zum Operateur sind seine Kenntnisse über die Einsatzsituationen und deren mögliche Folgen geringer und abstrakter. Mit der Umgebung Socially Augmented Microworld (SAM) ermöglicht ATEO einen Vergleich zwischen den Leistungen von Operateur und Entwickler.

Das System SAM wird von einem Operateur überwacht und geführt. Diese Versuchsumgebung basiert auf einem Trackingparadigma. Dabei führen zwei Versuchspersonen, Mikroweltbewohner (MWB) genannt, eine Trackingaufgabe durch. Die MWB steuern ein Objekt entlang eines Pfades vom Start bis ins Ziel. Für den Versuch bekommen beide unterschiedliche Ziele. Ein MWB soll die Strecke möglichst schnell absolvieren. Der andere soll das Objekt möglichst genau steuern. Die Steuerung des Fahrobjektes ist zu gleichen Teilen zwischen beiden MWB verteilt.

Die Trackingaufgabe wird durch den Operateur überwacht. Bei Bedarf kann dieser auch eingreifen. Der Operateur kann z.B. die Geschwindigkeit limitieren, den MWB Hinweise geben oder die Steuerungsanteile der einzelnen MWB ändern. Als ein Ergebnis seiner Diplomarbeit erstellte Hermann Schwarz [19] eine Machbarkeitsstudie des Operateursarbeitsplatzes. Dieses Graphical User Interface (GUI) soll verschiedene Eingriffe in das Tracking der MWB ermöglichen. Die Eingriffe werden in die Kategorien weich und hart unterteilt. Weiche Eingriffe sind Empfehlungen auditiver oder visueller Art. Diese sind für die MWB nicht verbindlich. Bei harten Eingriffen hingegen besteht keine Wahlmöglichkeit durch die MWB. Beispiele hierfür sind die Limitierung der Geschwindigkeit, eine Umverteilung der Steuerungsanteile oder das Sperren einer horizontalen Auslenkungsrichtung.

Dem Operateur werden durch den Operateursarbeitsplatz verschiedene Informationen bereitgestellt. Das sind z.B. eine Streckenvorschau, die Joystickauslenkungen der MWB oder die subjektiven Anstrengungsmaße. Das subjektive Anstrengungsmaß wird individuell für jeden MWB zwischen den Fahrten mit Hilfe einer dritten Softwarekomponente Mentale Anstrengung (MA) erfasst.

1.2 ZIELSTELLUNG

Ziel der vorliegenden Arbeit ist die Erstellung der Softwarekomponente Operateursarbeitsplatz, die Anbindung der Trackingkomponente SAM und die Komponente zur Erfassung der men-

talen Beanspruchung sowie die Implementierung der erforderlichen Anpassungen an die beiden Komponenten. Diese Arbeiten sollen in der Programmiersprache Smalltalk und der Umgebung Squeak implementiert werden. In Abbildung 1 ist die Zielstellung grafisch dargestellt.

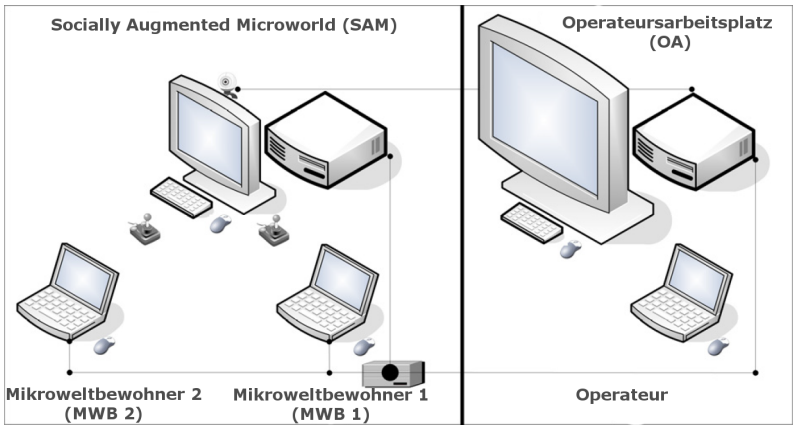

Abbildung 1: Versuchsaufbau des ATEO-Testsystems aus [15]

Der Operateursarbeitsplatz (OA) stellt dem Operateur Informationen über ein zeitgleich stattfindendes Tracking zweier MWB bereit. Der Operateursarbeitsplatz bietet darüber hinaus Eingriffsmöglichkeiten der Einflussnahme an. Diese Eingriffe sind die bereits beschriebenen weichen und harten Eingriffe.

Der Operateursarbeitsplatz besteht aus einer Reihe von Gui-Elementen. Jedes dieser Elemente lässt sich in eine von drei Kategorien zuordnen: Informationsanzeige, weiche und harte Eingriffe. Durch einen Konfigurationsdialog können verschiedene Varianten des Operateursarbeitsplatzes erstellt werden, in denen für jedes Element individuell entschieden wird, ob es angezeigt wird oder nicht. Aufgrund unterschiedlicher Versuchsanordnungen werden den Operateuren verschieden zusammengesetzte Operateursarbeitsplätze mit variierendem Informationsgehalt zur Verfügung gestellt.

Darüber hinaus bietet der Konfigurationsdialog die Möglichkeit, einige Eigenschaften der Elemente zu verändern. Damit kann beispielsweise festgelegt werden, ob sich die Elemente durch den Nutzer verschieben lassen oder der verwendete Mauszeiger verändert werden.

Die Kommunikation zwischen den einzelnen Komponenten SAM, Operateursarbeitsplatz und Mentale Anstrengung wird über ein Netzwerk erfolgen. Untersuchungen zu den erforderlichen Grundlagen erfolgten im Rahmen der Diplomarbeit von Nicolas Niestroj [18]. Die Umsetzung der Netzwerkkommunikation berührt alle vorher genannten Komponenten und ist Teil dieser Arbeit. Der Operateursarbeitsplatz wird als Teil dieses Kommunikationsnetzwerkes eine wichtige Funktion einnehmen

und als zentrale Anlaufstelle für die Netzwerkkommunikation der anderen beiden Komponenten agieren.

Wissenschaftlern aus dem ATEO Projekt des Graduiertenkollegs Prospektive Gestaltung von Mensch-Technik-Interaktion (prometei) soll es durch die Arbeiten ermöglicht werden, Versuche mit den Softwarekomponenten SAM, Operateursarbeitsplatz und Mentale Anstrengung durchzuführen und vorher aufgestellte Hypothesen zu prüfen.

1.3 VORARBEITEN IM PROJEKT ATEO

1.3.1 *Erste Umsetzung durch Maik Burandt*

In seiner Studienarbeit [5] wirkte Maik Burandt an der Erstellung der ersten Version des Operateursarbeitsplatzes mit. Damit wurde das Ende einer Entwicklungsstufe im Projekt ATEO erreicht.

Der Operateursarbeitsplatz war in diesem Entwurf nicht als Softwarekomponente mit einer Benutzeroberfläche konzipiert. Die Eingaben erfolgen über speziell angefertigte Tastaturen. Diese individuellen Tastaturen waren an den Versuchsrechner angeschlossen, auf dem auch die Trackingkomponente SAM lief. Die Hardware ist so gestaltet, dass jeder Tastendruck vom Rechner wie eine Eingabe einer beliebigen Tastatur interpretiert wird. Jede Taste ist einer eigenen Eingabe zugeordnet, um sie sinnvoll interpretieren zu können.

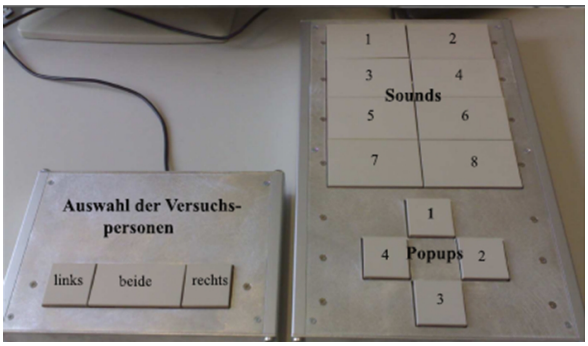

Abbildung 2: Tastatur des Operateurs

Die Operateurstastatur besteht aus zwei Teilen. Der kleinere Teil erlaubt die Auswahl von Adressaten von auditiven Hinweisen. Das können beide MWB sein oder auch nur ein einzelner MWB. Auf der anderen kann der Operateur einen der acht auditiven oder einen der vier visuellen Hinweise auslösen.

Die Hauptaufgabe von Maik Burandt war es, die Komponente SAM so anzupassen, dass sie die Eingaben der Operateurstastaturen verarbeiten kann. Dazu mussten vorhandene Klassen modifiziert und neue eingeführt werden.

Zur Verarbeitung der Eingaben der Tastaturen wurde die Klasse ATEOTastaturabfragen eingeführt. Die erkannten Eingaben werden von der ebenfalls neuen Klasse ATEOOperateur verarbeitet. In Abhängigkeit der Eingabe wurden die entsprechenden Aktionen, Abspielen von auditiven Hinweise oder Einblenden von visuellen Hinweisen, ausgelöst.

Eine weitere wichtige Aufgabe war die Erweiterung der Klasse ATEOLog. Das Logfile musste für spätere Auswertungen um die Informationen der neuen Funktionalitäten erweitert werden. Dazu wurden der Datei neue Spalten hinzugefügt, in denen vermerkt wurde, welche Hinweise zu welchem Zeitpunkt ausgelöst wurden. Im Falle der auditiven Hinweise wurde auch der Adressat in die Logdatei geschrieben.

Ein weiteres Thema der Studienarbeit war die Steigerung der Performance von SAM. Allerdings konnten in dieser Hinsicht keine nennenswerten Fortschritte erzielt werden. Darüber hinaus begann Maik Burandt mit der Kommentierung des Quellcodes.

Am Ende der Studienarbeit von Maik Burandt stand die Erkenntnis, dass SAM in Aspekten wie Architektur oder Performance weiterer Überarbeitungen bedarf.

1.3.2 *Vollständige Softwarelösung eines Operateursarbeitsplatzes von Herman Schwarz*

Geplante umfangreiche Überarbeitungen, Erweiterungwünsche und Limitierungen des bestehenden Operateursarbeitsplatz führten zum Entschluss, eine reine Softwarelösung zu erstellen. Die erste Studie dazu erstellte Herman Schwarz in seiner Diplomarbeit. In dieser Version wurden zwei wichtige Schritte vollzogen. Zum einen wurde die Verabeitung von Operateurseingaben aus der Komponente SAM herausgelöst. Herman Schwarz erstellte hierzu eine eigene Komponente für den Operateursarbeitsplatz. Als zweiten wichtigen Schritt ist das Ermitteln grundlegender Design Guidelines für den Operateursarbeitsplatz zu nennen. Die von Herman Schwarz geschaffenen Icons, Grafiken und Formen finden sich in der aktuellen Version wieder.

Eine Abbildung der Machbarkeitsstudie von Hermann Schwarz ist auf Seite 33 im Abschnitt 4.1 zu finden.

Durch die Aufspaltung der Softwarekomponente wurde es erforderlich einen Weg zu schaffen, über den beide Daten austauschen können. Die Versuchsanordnung bedingt zusätzlich eine räumliche Trennung des Operateurs und der MWB. Daher stehen die Rechner, auf denen die Software läuft, in verschiedenen Räumen. Nicolas Niestroj beschäftigte sich in seiner Diplomarbeit mit Möglichkeiten der Kommunikation in Netzwerken. Als Ergebnis identifizierte er Transmission Control Protocol / Internet Protocol (TCP/IP) und User Datagram Protocol (UDP) als mögliche Protokolle. TCP/IP kam überall dort zum Einsatz, wo eine sichere Übertragung im Vordergrund stand (Logging, Eingaben

des Operateurs). Übertragungen in hoher Frequenz, bei denen es keine signifikante Auswirkungen hat, wenn einzelne Pakete verloren gehen, wurden mit Hilfe des Protkolls UDP realisiert. Eine erste Implemtentierung entstand in Zusammenarbeit mit Herman Schwarz. In allen folgenden Versionen wurden die beiden Protokolle weiterhin eingesetzt. Die Implementierung wurde immer weiter vorangetrieben und gleichzeitig robuster gestaltet.

Die vorliegende Arbeit steht in direkter Linie zu den Vorarbeiten von Maik Burandt und Hermann Schwarz. Den größten Einfluß hat die direkte Vorarbeit von Hermann Schwarz [19]. In ihr sind grundlegende Fragestellungen zu Design wegweisend beantwortet worden. Die Resultate dieser Arbeit waren der Ausgangspunkt und die Basis der vorliegenden Arbeit. Bei der vorliegenden Arbeit handelt es sich jedoch um mehr als nur eine Überarbeitung vorhandener Ergebnisse. Vor allem die Dekomposition des Operateursarbeitsplatzes und die neu geschaffenen Elemente sind hier zu nennen. In der Version von Hermann Schwarz bestand der Operateursarbeitsplatz lediglich aus zwei einzelnen Teilen. Einige Elemente haben eine so grundlegende Neukonzeptionierung erfahren, dass sie sich sehr stark verändert haben. Beispiele dafür sind die Anzeige zur Joystickauslenkungen oder das Element zur Verteilung des Steuerungsanteils. Die Anzahl der zum Operateursarbeitsplatz gehörigen Klassen hat sich signifikant erhöht. Reichten in der Studie von Hermann Schwarz noch 18 Klassen aus, werden in der aktuellen Version 30 Klassen genutzt. Die aktuelle Version genügt den Ansprüchen der durchgeführten Versuche. Dennoch ist die Entwicklung noch nicht abgeschlossen. Künftige Versuche werden weitere Anpassungen und Neuentwicklungen am Operateursarbeitsplatz notwendig machen. Heutige Planungen zeigen dies bereits.

1.4 ERGEBNISSE UND AUFBAU

Am Beginn der Diplomarbeit wurde als Hauptziel die Schaffung eines einsatzbereiten Versuchssystems vereinbart. Jens Nachtwei sollte in der Lage sein, mit Hilfe dieses Versuchssystems psychologische Untersuchungen durchführen zu können. Dieses Ziel lässt sich in folgende Teilziele zerlegen:

- Erstellung eines funktionsfähigen Operateursarbeitsplatzes
- Vernetzung der Einzelkomponenten SAM, Operateursarbeitsplatz und Mentale Anstrengung

Alle gestellten Ziele wurden erreicht.

Wie schon bei den vorhergehenden Arbeiten kam als Softwareentwicklungsmethode Rapid Prototyping zum Einsatz. Die enge Zusammenarbeit mit Psychologen aus dem ATEO Projekt wurde

auf Basis agiler Prinzipien geführt. In der Projektarbeit bedeutete dies regelmässige, teils mehrfach tägliche Besprechungen. Diese Zusammenkünfte waren von einer zielorientierten und konstruktiven Atmosphäre geprägt. Als erster Schritt erfolgte eine Bewertung der Machbarkeitsstudie von Hermann Schwarz. Auf Basis dieser Erkenntnisse wurde eine grundlegende Architektur entworfen. Die praktischen Ergebnisse sind im Kapitel 4 dokumentiert. Nachdem nach zahlreichen Entwicklungszyklen eine erste vollständige Version des Operateursarbeitsplatzes vorlag, konzentrierten sich die weiteren Arbeiten auf die Vernetzung von SAM und Operateursarbeitsplatz. Maßgeblich für diesen Teil war die konzeptionelle Vorarbeit von Nicolas Niestroj [18] und die Implementierungen der vorhergehenden Studienarbeit [12]. Danach folgten weitere Entwicklungsiterationen, in denen der Operateursarbeitsplatz immer wieder überarbeitet und erweitert wurde. Die Ergebnisse werden in den Kapiteln 5 (Diskussion) und 6 (Ausblick) dargelegt.

2 GRUNDLAGEN

2.1 OBJEKTORIENTIERTE PROGRAMMIERUNG MIT SQUEAK UND SMALLTALK

Squeak ist eine quelloffene Implementierung von Smalltalk. Die erste Version wurde von Alan Kay, Ted Kaehler, John Maloney, Scott Wallace und Dan Ingalls entwickelt. Diese Gruppe hatte bereits entscheidenden Anteil an der Smalltalk-80 Umsetzung.

Die erste Version von Squeak wurde 1996 veröffentlicht. Eine Hauptmotivation war es, eine Umgebung zu schaffen, in der sich auch Menschen mit weniger technischen Kenntnissen dem Thema Programmierung nähern können. Vorher formulierte Eigenschaften waren unter anderem einfaches, flexibles Design und eine effiziente Nutzung von Speicher und Rechenleistung. Vor allem das Design von Squeak sollte rasche Anpassungen und Erweiterungen ermöglichen.

In den vergangenen 16 Jahren wurde Squeak kontinuierlich weiterentwickelt und liegt aktuell in der Version 4.3 vor.

Die eingesetzte Sprache Smalltalk und die Entwicklungsumgebung Squeak verfügen über eine Reihe wichtiger Eigenschaften:

- Alles in Squeak ist ein Objekt.
- Alle Methoden in Smalltalk sind öffentlich und die Attribute sind private.
- In Smalltalk wird dynamisch getypt.
- Smalltalk nutzt einfache Vererbung und Polymorphismus.
- Squeak nutzt keine Sourcecodedateien im Filesystem, sondern speichert den Quellcode in einem Image.
- Es existiert keine explizite Kompilierung oder Verlinkung.
- Darüber hinaus kommt wie in C# oder Java eine Virtual Machine zum Einsatz. Wie auch in diesen beiden Sprachen wird Garbage Collection eingesetzt. Das heisst, der Benutzer hat keinen Einfluss auf die Freigabe oder Reservierung von Speicher.

Die Umgebung Squeak mit seinen Werkzeugen, z.B. dem Klassen- und Prozessbrowser, Workspaces oder Methodfinder, sind komplett in Smalltalk programmiert. Daher ist es sehr einfach möglich, an diesen Squeak-Elementen Änderungen vorzunehmen.

2.2 VERNETZTE ANWENDUNGEN

2.2.1 *Architekturen für verteilte Systeme*

Das ATEO-Versuchssystem setzt sich aus einer Reihe durch ein Netz verbundener Softwarekomponenten zusammen. Ein derartiges System wird von Balzert [1] als ein verteiltes System bezeichnet. Der Aufbau verteilter Systeme folgt einem Architekturparadigma. Für die Klassifizierung sind Merkmale wie das Verhältnis der Einzelkomponenten zueinander oder die Art der Bereitstellung der Dienste wesentlich. Komponenten, die Dienste anbieten, werden Server genannt. Komponenten, die Dienste in Anspruch nehmen, heissen dagegen Clients. Balzert [1] nennt unter anderem folgende Architekturen:

2.2.1.1 *Client-Server-Architektur*

Bei dieser Architektur werden die Subsysteme auf wenigstens einen Server (Backend) und wenigstens einen Client (Frontend) verteilt. Der Server stellt einen oder mehrere Dienste bereit. Die Clients sind Nutzer dieser Dienste. Sie sind auch immer die Initiatoren der Kommunikation. Allerdings entscheidet der Server selbst über die Priorisierung der einzelnen Anfragen. Beispielsweise muss ein Datenbankserver sicherstellen, dass verschiedene Anfragen, welche auf die gleichen Objekte zugreifen, sicher und konsistent durchgeführt werden. Für Clients sind potentielle Konflikte z.B. durch die gleichzeitige Nutzung kritischer Ressourcen nicht zu erkennen. Je nach Ausgestaltung unterscheidet man mehrere Arten von Clients. Die beiden wichtigsten sind der Fat-Client und der Thin-Client. Ein Thin-Client verfügt über eingeschränkte Hardwarekapazitäten, da die meisten Aufgaben auf dem Server ausgeführt werden. Normalerweise ist auf einem Thin-Client nur ein Betriebssystem installiert. Das bedeutet aber auch, dass Applikationen nicht ohne Netzwerkanbindung zu nutzen sind. Um die fehlende Leistung der Clients zu kompensieren, muss der Server leistungsfähiger sein. Der Fat-Client hingegen verfügt über leistungsfähigere Hardware, da Teile der Anwendung auf dem Rechner selbst durchgeführt werden. Das entlastet wiederum den Server. Durch Ausführung einzelner Prozessschritte auf dem Client ist eine permanente Verbindung zum Server nicht erforderlich. Trotzdem muss eine Verbindung zum Server regelmäßig hergestellt werden.

Beispiele:

- eMail Systeme mit Clients wie MS Outlook in Unternehmen
- Active Directory Service (Benutzerverwaltung)

2.2.1.2 *Web-Architektur*

Hierbei handelt es sich grundsätzlich um eine Client-Server-Architektur, bei der Webtechnologien Verwendung finden. Für die Kommunikation zwischen Server und Client wird das HTTP Protokoll eingesetzt. Die Darstellung der GUI erfolgt mit Hilfe eines Webbrowsers. Die Verarbeitung der Daten erfolgt zum grössten Teil auf dem Server. Technologien wie JavaScript oder HTML5 ermöglichen eine eingeschränkte Datenverarbeitung auch auf dem Client. Trotzdem ist der Server der begrenzende Faktor des Gesamtsystems, da er den überwiegenden Teil der Datenverarbeitung leistet. Im Regelfall übernimmt er sogar die komplette Verarbeitung. Vorteil dieser Architektur ist es, dass auf dem Client nur ein Webbrowser installiert sein muss. Das Betriebssystem ist damit von nachgeordneter Relevanz.

Beispiele:

- GUIs von Netzwerkgeräten wie Router, Access Points (AP) oder Networked Attached Storage (NAS)

2.2.1.3 *Serviceorientierte Architekturen*

Bei dieser Architektur steht die Bereitstellung und Kombination von Funktionalitäten im Vordergrund. Durch eine Serviceorientierte Architektur soll es ermöglicht werden, aus einzelnen Funktionen neue Anwendungen zu erstellen. Die Programmiersprache einer Implementierung einer Funktion ist dabei nebensächlich. Kommunikation mit einer Funktion erfolgt hier ausschließlich über standardisierte, selbstbeschreibende und veröffentlichte Schnittstellen. Die Veröffentlichung erfolgt in einem Verzeichnis. Damit können verschiedene Technologien parallel eingesetzt werden. Insbesondere für Altsoftware, sogenannte Legacy-Software, stellt das eine Möglichkeit dar, deren Funktionalitätem weiter zu nutzen, ohne die Anwendungen neu implementieren zu müssen.

Beispiel:

- Webservices von Amazon

2.2.1.4 *Peer-To-Peer-Architektur*

Das Hauptmerkmal dieser Architektur ist die Gleichberechtigung der Einzelkomponenten. Alle beteiligten Komponenten stellen hierbei gleichermaßen Dienste bereit und nehmen sie in Anspruch. Sie sind demzufolge gleichzeitig Server und Client. Es existiert auch keine zentrale Infrastruktur. Jeder Teilnehmer ist mit jedem anderen verbunden. Meist kann in P2P Anwendungen keine verlässliche Aussage über Bandbreite, Verfügbarkeit oder Leistungsfähigkeit der Teilnehmer getroffen werden.

Beispiele:

- Facetime (Videotelefonie)
- Bittorrent (Filesharing)

Die Zuordnung des ATEO Versuchssystem zu einer der vorgestellten Architekturen ist schwierig, da es Merkmale verschiedener Architekturen aufweist. Betrachtet man den Operateursarbeitplatz und die beiden Mental-Workload Komponenten, so ist dieser Teil der Client-Server Architektur zuzuordnen. Andererseits ähnelt die Beziehung von SAM und dem Operateursarbeitplatz eher der von Clients in einer P2P Architektur. Insgesamt handelt es sich hier also um eine Mischform. Sie ist das Resultat der fachlichen Anforderungen an das Versuchssystem.

2.2.2 *Möglichkeiten der Kommunikation in einem Netzwerk*

Die einzelnen Teile des ATEO Versuchssystems sind über ein Netzwerk miteinander verbunden. Für die Kommunikation zwischen den Komponenten stehen verschiedene Möglichkeiten zur Vefügung. Beispiele:

- SOAP
- Sockets
- CORBA
- Google Protocol Buffers

Für die Implementierung der Netzwerkkommunikation für das Versuchssystem wurden Sockets verwendet. Wichtige Gründe für die Entscheidung war die damit verbundene Flexibilität und einfache Handhabung. Allerdings stellen Sockets nur eine grundlegende Funktionalität zur Verfügung. Funktionalitäten wie Verbindungsaufbau und -abbau oder Kommunikationsprotokolle mussten erst entworfen und implementiert werden.

2.2.2.1 *Stream Sockets*

Für Netzwerkkommunikation, deren Schwerpunkt auf Zuverlässigkeit lag und Geschwindigkeit gleichzeitig vernachlässigt werden konnte, wurden Stream Sockets eingesetzt. Diese Sockets verwenden Zeichenströme für die Kommunikation. Stream Sockets verwenden in der Smalltalk Implementierung als Übertragungsprotokoll TCP. TCP ist ein verbindungsorientiertes Protokoll, bei dem Sender und Empfänger ständig in Kontakt stehen. Das Protokoll verfügt über eine Fehlerbehandlung. Fehler werden durch einen kontinuierlichen Austausch von Kontrollmeldungen festgestellt. Verlorene Pakete können somit erkannt und erneut angefordert werden. Auch die korrekte Reihenfolge wird durch

TCP auf Empfängerseite sichergestellt. Ein weiterer Vorteil ist die Flusskontrolle. Mit ihrer Hilfe kann die Datenrate an die zur Verfügung stehende Bandbreite angepasst werden.

2.2.2.2 *Datagram Sockets*

Bei Netzwerkkommunikation, die primär eine schnelle Übertragung der Informationen erfordert, wurden Datagram Sockets eingesetzt. Im Gegensatz zu Stream Sockets werden hier keine Zeichenströme sondern Nachrichten versandt. Datagram Sockets verwenden als Protokoll UDP. Dieses ist ein verbindungsloses Protokoll, welches die gleiche Aufgabe wie TCP erfüllt. Allerdings fehlen ihm weitestgehend die Kontrollmechanismen von TCP. Der Sender kann nicht feststellen, ob seine Nachricht angekommen ist. Da ein Verlust hier weder bemerkt noch kompensiert wird, ist dieses Protokoll unsicherer. Auch die Reihenfolge der Nachrichten ist nicht implizit sichergestellt, da für den Empfänger nicht zu erkennen ist, in welcher Reihenfolge die Nachrichten gesendet wurden. Dieses Feature muss individuell implementiert werden, z.B. durch eine fortlaufende Nummer als Teil der Nachricht. Angemessen ist der Einsatz von UDP dann, wenn es primär um die Geschwindigkeit der Übertragung geht und es gleichzeitig zu tolerieren ist, dass einzelne Nachrichten verloren gehen. Beim Audio- oder Videostreaming ist das der Fall. Im ATEO Versuchssystem wurden Datagram Sockets eingesetzt, um die Daten des Trackings von SAM an den Operateursarbeitplatz zu übertragen.

3 THEORIE

3.1 ANWENDUNGSGEBIET: DESIGN EINES OPERATEURSARBEITSPLATZES

Das gesamte Projekt ATEO und damit auch diese Diplomarbeit, ist dem Bereich Human Factors zuzuordnen. Dieser Bereich wird in der Literatur manchmal auch als Ergonomie bezeichnet. Human Factors ist ein wissenschaftliches Feld, welches Einflüsse aus verschiedensten Dispziplinen in sich vereinigt, beipielsweise Psychologie, Ingenieurswesen und Design. Die International Ergonomics Association (IEA) definiert Human Factors wie folgt:

> „Ergonomie (oder Human Factors) ist die wissenschaftliche Disziplin, die sich mit dem Verständnis von Interaktion zwischen Menschen und anderen Elementen eines Systems beschäftigt und die praktische Anwendung welche die Theorien, Prinzipien, Daten und Methoden in Bezug auf Design nutzt, um menschliches Wohlergehen und Systemleistung zu optimieren."

Human Factors hat zum Ziel, den Entwurfsprozess von Produkten zu untersuchen und zu verbessern. Damit sollen Produkte geschaffen werden, die optimal an die körperlichen und kognitiven Möglichkeiten des menschlichen Körpers angepasst sind. In der Psychologie thematisiert die Ingenieurspsychologie Ziele und Probleme von Human Factors. In der Informatik übernimmt das die Softwareergonomie. Auch sie setzt sich mit den Fragestellungen aus dem Bereich Human Factors auseinander. Der Anwendungsbereich von Human Factors ist sehr umfangreich und berührt die Entwurfsprozesse von z.B. Haushaltsgeräten, Werkzeugen oder Software gleichermaßen. Die IEA teilt Human Factors in verschiedene Bereiche ein:

- Physische Ergonomie: Dieser Bereich beschäftigt sich vorrangig mit dem körperlichen Aspekten wie Anatomie, Physiologie oder Biomechanik. Untersucht werden hier z.B. die Körperhaltung beim Arbeiten, Arbeitsplatzdesign, sich wiederholende Tätigkeiten oder Sicherheit.
- Kognitive Ergonomie: Hier liegt der Schwerpunkt auf den mentalen Prozessen wie Wahrnehmung, Erinnerung, Schlussfolgerung und motorische Reaktionen. Hier werden Themen betrachtet, die im Zusammenhang mit dem Design

von Mensch-Maschine Systeme stehen. Das sind u. a. Mentale Belastung, Mensch-Maschine-Interaktion oder Entscheidungsprozesse. Die vorliegende Diplomarbeit gehört zu diesem Teilbereich von Human Factors.

- Organisatorische Ergonomie: Hier steht die Optimierung von soziotechnischen Systemen im Mittelpunkt. Das umfasst deren Organsationsstrukturen, Richtlinien und Prozesse. Schwerpunkte sind Kommunikation, Teamwork, Organisationskultur oder Qualitätsmanagement.

Die in Kapitel 3.3 beschriebenen Verfahren Hierarchische Task Analyse, Usability Test und Heuristische Evaluation sind typische Methoden aus dem Bereich Human Factors. Gleiches gilt auch für die eingesetzte Softwareentwicklungsmethode Rapid Prototyping. Die IEA nennt in der Definition für Human Factors „...menschliches Wohlergehen und Systemleistung...“ als Ziele. Diese werden im Folgenden erläutert.

MENSCHLICHES WOHLERGEHEN Die Operateure der durchgeführten Usability Tests wurden nach Abschluss der Versuche mit Hilfe eines Fragebogens nach ihrer subjektiven Zufriedenheit in Bezug auf den Operateursarbeitsplatz befragt. Zusätzlich wurde in den Versuchen die mentale Belastung ermittelt. Diese sollte weder zu niedrig noch zu hoch sein. Beide Extremzustände wirken sich nachteilig auf das Wohlbefinden und die Leistung des Operateurs aus. Die Anwendung der in der Diplomarbeit von Hermann Schwarz [19] ermittelten Styleguides diente ebenfalls dem Ziel, dem Operateur ein bestmöglich gestalteten Operateursarbeitsplatz bereitzustellen und somit die Zufriedenheit zu erhöhen.

SYSTEMLEISTUNG Um die optimale Systemleistung zu erzielen, wurden die Aufgaben mit Hilfe der Hierarchical Task Analysis (HTA) analysiert und beschrieben. Damit sollte ein möglichst umfassendes Verständnis über die Tätigkeiten des Operateurs geschaffen werden. Eine frühe Umsetzung im Operateursarbeitsplatz wurde danach von Experten evaluiert. In den Usability Tests wurden mit Hilfe von Kriterien wie Genauigkeit und Geschwindigkeit des Trackings die Systemleistung überprüft. Aus den gesammelten Erkenntnissen wurden Verbesserungen abgeleitet. Diese wurden konsequent umgesetzt, um eine Steigerung der Systemleistung zu erzielen. Darüber hinaus wurde das Situationsbewusstsein der Operateure untersucht. Dieses sollte so hoch wie möglich sein, denn nur, wenn ein Operateur optimal über den Systemzustand informiert ist, kann er hohe Leistungen erbringen.

3.2 EINORDNUNG IN SOFTWARE DEVELOPMENT LIFCYCLE

Zur Erstellung von Software steht eine Anzahl von Vorgehensweisen zur Verfügung. Die einzelnen Methoden unterscheiden sich in Merkmalen wie Strukturierung (stark strukturiert vs. flexibel und offen) oder Perspektive (Top-Down vs. Bottom Up). Zu Beginn eines Projektes ist eine Methode auszuwählen, die am besten zu den Rahmenbedingungen passt. Eine wesentliche Rahmenbedingung für die vorliegende Diplomarbeit war die noch nicht finale Spezifikation. Bereits vor Beginn der Arbeit war bekannt, dass sich die existierende Spezifikation noch signifikant ändern würde. Aus diesem Grund wurde eine Vorgehensweise gewählt, die diesem Umstand angemessenen Platz einräumt. Ein stark strukturiertes Vorgehensmodell wie das Wasserfallmodell, Spiralmodell oder das das von IBM entwickelte Rational Unified Process Modell (RUP) wären für eine Aufgabe mit einer deartigen Einschränkung nur bedingt geeignet. Diese Methoden sind eher für Vorhaben nutzbar, die aufgrund einer vollständigen Spezifikation eine bessere Planbarkeit erlauben. Die daher in Frage kommende Vorgehensweise sollte so flexibel sein, dass sie Anpassungen der Spezifikation zur Projektlaufzeit ermöglicht. Das kann beispielsweise notwendig werden, wenn sich Anforderungen des Auftraggebers ändern oder die Implementierung der Spezifikationen sich zu aufwendig oder als nicht umsetzbar erweisen. Auch wenn das Zielsystem bei Projektbeginn nicht vollständig in Umfang und Verhalten beschreibbar ist, sollte durch das Vorgehensmodell Änderungen Platz eingeräumt werden. Bei der vorliegenden Arbeit wurde Evolutionary Prototyping als Methode eingesetzt. Sie ist der Familie der Prototyping Methoden zuzuordnen. Bei dieser Methodenfamilie steht eine zügige Erstellung eines Prototyps im Fokus. Das kann ein ganzes System oder ein Teil davon sein. Ein Prototyp kann als Diskussionsgrundlage dienen, sowohl mit einem Auftraggeber oder innerhalb eines Entwicklerteams. Das gemeinsame Verständnis aller Beteiligten über das Ziel und das Vorgehen wird gestärkt. Zusätzlich können anhand eines Prototyps Probleme verdeutlicht, Lösungsvorschläge erarbeitet und Entscheidungen herbeigeführt werden. Die erstellten Prototypen dienten nicht nur dem Ziel ein Design zu erarbeiten. Reine Designprototypen hätten auch mit einfacheren Methoden erstellt werden können. Im vorliegenden Fall lag der Fokus auf dem Zusammenspiel mit der bereits vorhandenen Trackingkomponente SAM und dem Verhalten des Operateursarbeitsplatzes bei der Ausführung. Ein weiterers wichtiges Merkmal von Prototyping ist die Flexibilität gegenüber Anforderungsanpassungen zur Projektlaufzeit. Ein Prototyp ist gewissermassen eine ausführbare Spezifikation. Damit ist die ausgewählte Methode im hohen Maße für die bearbeitete Aufgabe geeignet. Eine weitere Prototyping

Methode ist Exploratives Prototyping. Sie hat das Ziel, eine bessere und übersichtlichere Spezifikation zu entwickeln. Daneben existiert u.a. noch das Experimentelle Prototyping. Hiermit lassen sich potentielle Implementierungmöglichkeiten bestimmen. Das gewählte Vorgehensmodell wurde um eine Reihe von Prinzipien aus dem „Manifest für Agile Softwareentwicklung“ erweitert. Sie beschreiben beispielsweise die Gestaltung von Zusammenarbeit mit Auftraggebern, die Organisation von Teams oder die Strukturierung von Informationsflüssen. Die 12 Prinzipien sind im Detail im Anhang im Kapitel A.4 auf Seite 95 zu finden. Die Methode des Prototypings gab den Rahmen und die Reihenfolge der einzelnen Prozessschritte vor. Die agilen Prinzipien bestimmten WIE diese Schritte vollzogen wurden.

3.2.1 *Evolutionary Prototyping in der Anwendung*

Das hier verwendete Evolutionary Prototyping eignet sich in besonderem Maße für die Aufgabe. In kurzer Zeit wurde ein Prototyp erstellt, der in folgenden Iterationen immer weiter überarbeitet wurde. Diese Eigenschaft des Vorgehens passt sehr gut zur hohen Dynamik der Anforderungen des Operateursarbeitsplatzes. Weiterhin schaffte dieses Vorgehen eine gute Grundlage für den interdisziplinären Charakter der Arbeit. Die Vorgaben wurden von Psychologen erstellt und von Informatikern umgesetzt. Als Vorlage für den Operateursarbeitsplatz diente eine Skizze aus Microsoft Powerpoint. Darauf waren zwar die Einzelelemente und ihre Anordnungen zu erkennen, jedoch nicht ihre Funktionen. Auch das individuelle Verhalten der Einzelelemente zur Laufzeit war nicht erkennbar. Die Wünsche und Erwartungen wurden in regelmäßig stattfindenden Workshops herausgearbeitet. Hier zeigte sich einer der Vorteile des Prototypings, da innerhalb einer relativ kurzen Zeit eine erste Version zur Verfügung stand. Anhand dieses Prototypen konnten Unklarheiten beseitigt und Veränderungswünsche erläutert werden. Die Ergebnisse wurden daraufhin in einer neuen Skizze dokumentiert. Danach wurde die neue Vorlage in den Prototypen eingearbeitet und dieser erneut bewertet. Durch dieses Vorgehen wurde der Operateursarbeitsplatz evolutionär entwickelt.

Einige Beispiele der verwendeten Vorlagen sind im Anhang im Abschnitt A.2 ab Seite 89 zu finden.

Prototyping im Allgemeinen lässt sich in vier Prozessschritte aufteilen:

1. Identifizierung der grundlegenden Anforderungen - in diesem Schritt werden grob die Anforderungen herausgearbeitet. Die Bezeichnung grob bezieht sich hierbei auf die wichtigsten Eigenschaften. Dabei wird nach einem Top Down Ansatz verfahren. Das bedeutet, dass in der initialen Spezifikation nur die für den Anwender wichtigsten, charak-

teristischsten Merkmale festgehalten werden. Im Falle des Operateursarbeitsplatzes wurde ein Element erst einmal nur in ungefährer Größe, Anzahl von Einzelelementen, wie Buttons oder Schieberegler, und deren annähernden Positionierung beschrieben. Details wie Schriftarten, die genaue Farbe oder die pixelgenaue Anordnung waren nicht Teil dieser Spezifikation. Für die meisten Elemente wurde zunächst auch kein Verhalten definiert.

2. Entwicklung des Prototypen - Auf Basis der in Schritt eins gefundenen Beschreibung wird ein erster Entwurf erstellt. Im Normalfall enthält dieser Prototyp noch kein Verhalten, d.h. Elemente wie Buttons werden ohne Funktionalität implementiert. Im Entwurfsprozess des Operateursarbeitsplatzes wurde in diesem Schritt die Klasse OpWindowRectangleMorph in ihrer ersten Version implementiert. Diese Klasse dient als Template für die Einzelelemente des Operateursarbeitsplatzes.

Die Klasse OpWindowRectangleMorph wird in Kapitel 4.3.1.1 auf Seite 43 beschrieben.

3. Review - Der Prototyp aus Schritt zwei wird dem Kunden vorgestellt und gemeinsam besprochen. Da es sich in der ersten Iteration um eine unvollständige Implementierung handelt, entstehen bei der Untersuchung durch den Kunden mit hoher Wahrscheinlichkeit weitere Anpassungswünsche am Prototypen. Das können einerseits Änderungen bereits umgesetzter Teile oder Neuerungen sein. Darüber hinaus werden an dieser Stelle im Prozess vorher unspezifizierte Aspekte konkretisiert (z.B. Schriftart oder Farbe) oder neue Anforderungen formuliert. Die unter 3.3.2 Heuristische Evaluation und die unter 3.3.3 Usability Test beschriebenen Vorgehen waren Reviewschritte im durchgeführten Prototypingprozess.

4. Neubewertung und Überarbeitung des Prototypen - Die aus Schritt drei gewonnenen Erkenntnisse werden im vierten Schritt umgesetzt. Das bedeutet, dass die Spezifikation erweitert oder korrigiert wird. Die Implementierung wird im Gleichschritt mit der Spezifikation ebenfalls angepasst. Nach der Umsetzung der Änderungen werden die Schritte drei und vier wiederholt. Der Entwicklungsprozess ist dann abgeschlossen, wenn in Schritt drei keine neuen Änderungswünsche herausgearbeitet werden. Für den Operateursarbeitsplatz wurden als Ergebnis von Schritt drei neue Vorlagen erstellt. Auf Basis dieser Vorlagen fanden Anpassungen an der Implementierung statt. Änderungen oder Erweiterungen am Verhalten des Operateursarbeitsplatzes wurden in weiteren Workshops erarbeitet.

Die Prozessschritte drei und vier wurden in relativ kurzen Zyklen durchlaufen. Dabei reichte die Dauer von etwa einem Tag bis hin zu zwei Wochen je nach Komplexität der Änderungen. Dabei konzentrierte sich die Entwicklung nicht immer auf den gesamten Operateursarbeitsplatz, phasenweise wurden der Fokus auf einzelne Elemente gelegt. Auf diese Weise wurde die komplexen Elemente, wie die Streckenvorschau, umgesetzt. Eine Besonderheit bei dieser Arbeit war, dass die tatsächlichen Endnutzer nicht am Entwicklungsprozess teilnahmen. Die potentiellen Benutzer des Operateursarbeitsplatzes, die Operateure, waren Versuchspersonen, die während der Entwicklung noch nicht bestimmt waren. Darüber hinaus hätte die Involvierung bei der Entwicklung sie für den Einsatz in den Versuchen unbrauchbar gemacht. Die Hauptansprechpartner und damit die hauptsächlichen Reviewer waren die am Projekt teilnehmenden Psychologen

3.2.2 *Horizontales und Vertikales Prototyping*

Softwareentwicklung kann auch als Entwurf und Implementierung einer Anzahl von Schichten verstanden werden. Diese Schichten können z.B. das User Interface oder das Betriebssystem sein. Vor diesem Hintergrund kann man in horizontale und vertikale Prototypen unterteilen.

1. Ein horizontaler Prototyp implementiert nur eine Schicht des Zielsystems. Das kann zum Beispiel das komplette User Interface mit seinen Fenstern und Menüs eines Zielsystems sein.

2. Der vertikale Prototyp realisiert einen ausgewählten Teil des Zielsystems durch alle Schichten hindurch. Das könnte beispielsweise eine einzelne Komponente der GUI mit ihrer gesamten Funktionalität sein.

Im Verlauf der Arbeiten am Operateursarbeitsplatz wurde zuerst, in Prozessschritte zwei aus dem in 3.2.1 erläuterten Vorgehen, ein horizontaler Prototyp erstellt. In ihm wurde ausschließlich das User Interface ohne Verhalten implementiert. Die Einzelelemente des Operateursarbeitsplatzes wurden zum überwiegenden Teil - Schritt vier aus 3.2.1 - als vertikale Prototypen umgesetzt. Für den gesamten für das Element relevanten Bereich wurde eine Implementierung durch alle Schichten vorgenommen. Beispielsweise wurde für das Element „Auditive Hinweise“ die Logik der Buttons, die Datenspeicherung und die Netzwerkfunktionalität gemeinsam umgesetzt. Aufgrund der Beziehungen der Elemente untereinander konnten manche vertikale Prototypen nur Zug um Zug realisiert werden. Davon betroffen waren z.B. die Elemente,

deren Zustände auch in der Streckenvorschau dargestellt werden. Das sind u. a. die Elemente Geschwindigkeitslimit, Steuerungsanteil oder Richtungsbeschränkung. Erst wenn alle beteiligten Elemente einen gewissen Reifegrad haben, können alle Implementierungen vorgenommen werden.

3.2.3 *Agile Prinzipien*

Klassische Vorgehensmodelle sind mitunter in ihrer praktischen Anwendung formal und bürokratisch. Einen Gegenentwurf dazu bilden die sogenannten Agilen Prinzipien. Im Jahr 2001 formulierte eine Gruppe von Informatikern das „Manifest für Agile Softwareentwicklung“ [3]. Die Basis bilden folgende vier Kernaussagen:

1. Menschen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
2. Funktionierende Software ist wichtiger als umfassende Dokumentation.
3. Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.
4. Eingehen auf Veränderungen ist wichtiger als Festhalten an einem Plan.

Zu diesen Aussagen gehört noch folgender Satz:

> "Das heißt: Obwohl die Punkte auf der rechten Seite [der Aussagen] durchaus wichtig sind, halten wir die Punkte links für wichtiger".

Diese Feststellungen basieren auf der Erkenntnis, dass allein die Einhaltung von Regeln des eingesetzten Vorgehensmodells nicht den Erfolg garantiert oder ihn manchmal sogar verhindert. Gerade Projekte mit dynamischen Anforderungen können von diesen Prinzipien profitieren, da sie den Umgang mit „Richtungsänderungen“ berücksichtigen. Die Rahmenbedingungen bei dieser Diplomarbeit legten den Einsatz von agilen Prinzipien nahe. Weiterhin wurden agile Prinzipien bereits in anderen Arbeiten im ATEO Umfeld mit großem Erfolg eingesetzt. Im Folgenden werden die vier Kernaussagen und ihre Anwendung auf die Entwicklung des Operateursarbeitsplatzes dargestellt.

3.2.3.1 *Menschen und Interaktionen sind wichtiger als Prozesse und Werkzeuge*

Das Kernteam im Projekt in Bezug auf das Diplomthema bildeten zwei Personen. Phasenweise erhöhte sich die Anzahl auf bis zu

vier Personen. Zusätzlich konnten die Arbeiten in räumlicher Nähe stattfinden. Somit war keine aufwendige Projektorganisation erforderlich. Daher konnte auf ein dediziertes Projektmanagementoffice (PMO) zur Koordinierung von Ressourcen verzichtet werden. Im Umkehrschluss bedeutete dies nicht, dass klare Strukturen und Prozesse unnötig waren. Sie wurden hinsichtlich Aufwand und Komplexität angemessen gestaltet. So folgte z.B. die Erstellung einer neuen Version eines Elementes des Operateursarbeitsplatzes einem festgelegten Prozess:

1. In einem ersten Schritt wurden mögliche Anpassungen erarbeitet. Das erfolgte gemeinsam mit dem Projektteam in Workshops, auf Basis von Fachliteratur oder durch Expertengespräche.

2. In Schritt zwei wurden diese Ideen in eine neue Version der Abbildung des Operateursarbeitsplatz eingefügt und danach an die umsetzenden Personen weitergeleitet.

3. Basierend auf dieser aktualisierten Abbildung des Operateursarbeitsplatzes und ggf. begleitenden Gesprächen mit der Fachseite wurden unklare Sachverhalte ausgeräumt und dann die Umsetzung begonnen.

4. Nach der Umsetzung folgten die Abnahme und Tests. Im Anschluß daran wurden die eventuell gefundenen Fehler behoben.

Die Methode Heuristische Evaluation (Expertengespräche) wird in Kapitel 3.3.2 auf Seite 25 beschrieben.

Zusätzlich wurden auch Meilensteine vereinbart und deren Fortschritt überwacht. Während der vorliegenden Arbeit wurden die gefundenen Fehler in einer zentralen Datei gesammelt und deren Bearbeitungsstand verfolgt.

3.2.3.2 *Funktionierende Software ist wichtiger als umfassende Dokumentation*

Es steht ausser Frage, dass eine umfassende Dokumentation wünschenswert ist. Wenn Dokumentation und Fertigstellung in Konflikt geraten, sollte jedoch der Fertigstellung der Software Vorzug gegeben werden. Gründe für eine derartige Situation könnten fehlende Ressourcen oder unzureichende Planung sein. Die Arbeiten am Operateursarbeitsplatz wurden nicht so umfangreich und detailliert dokumentiert, wie es aufgrund der Komplexität angemessen gewesen wäre. Aufgrund von Rahmenbedingungen konkurierten beide Aufgaben um die knappe Ressource Zeit. Die Durchführung der Versuche mit dem Operateursarbeitsplatz waren ebenfalls einem Zeitplan unterworfen, der nur mit erheblichen Kosten hätte angepasst werden können. Daher hatte die

Einsatzfähigkeit des Operateursarbeitsplatzes Priorität. Die Dokumentation wurde erst im Anschluss durch die Erstellung dieser Arbeit nachgeholt.

3.2.3.3 *Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen*

Die Aussage lässt sich auf den ersten Blick nicht vollständig auf diese Arbeit anwenden. Sie basiert auf Projekterfahrungen aus der Praxis, die zeigen, dass aus Sorge vor Kostenexplosion oder verspäteten Projektabschluss an den ursprünglichen Spezifikationen und Planungen festgehalten wird. Auch wenn diese durch Projektfortschritt überholt und damit hinfällig sind. Obwohl dieser Arbeit die wirtschaftliche Komponente fehlte, lässt sich die Aussage anwenden. Der Fokus liegt dabei eher auf dem ersten Teil. Kooperation bei Problemen und nicht das Zurückziehen auf bereits vereinbarte Ziele sollten das Handeln bestimmen. Das betrifft vor allem die aufwändigen Fälle, welche z.B. fertige Implementierungen obsolet machen und neue erfordern. Einige Elemente im Operateursarbeitsplatz unterliefen in ihrer Evolution zum Teil drastische Änderungen. Ohne die Kooperation der Fachseite und der Implementierer wäre die Umsetzung bestenfalls problematisch geworden.

3.2.3.4 *Eingehen auf Veränderungen ist wichtiger als Festhalten an einem Plan*

Im Normalfall werden Softwareprojekte vor Beginn weitestgehend durchgeplant und Aufwands- und Kostenschätzungen vorgenommen. Treten nach dem Projektstart unvorhergesehene Situationen auf, die im Vorfeld nicht planbar oder vorhersehbar waren, ist es wichtig diese im weiteren Vorgehen zu berücksichtigen und nicht zu ignorieren. Das Projekt um den Operateursarbeitsplatz war nur eingeschränkt planbar. Es war nicht absehbar, welche Impulse es beispielsweise durch die Heuristische Evaluation geben würde. Andererseits konnten einige Ideen erst durch eine Implementierung hinreichend geprüft werden. Andere entwickelten sich erst im Laufe der Zeit ausreichend weit, damit sie umgesetzt werden konnten. Zusätzlich beeinflussten sich die Elemente gegenseitig, so dass Änderungen weitere Änderungen auslösten. Es war bei Beginn der Arbeit nicht annähernd möglich, das System in Umfang und Verhalten komplett zu beschreiben. Die Flexibilität beim Vorgehen war eine Grundvorraussetzung für die erfolgreiche Durchführung der Arbeit.

Die Kombination von Evolutionary Prototyping und agilen Prinzipien hat sehr gut funktioniert. Dies wurde durch die relativ geringe Anzahl der beteiligten Personen begünstigt. Zusätzlich wurde durch die Bereitstellung von Arbeitsplätzen am psycholo-

gischen Lehrstuhl die Kommunikation deutlich vereinfacht und Wege verkürzt.

3.3 MÖGLICHKEITEN DES USABILITY ENGINEERING ZUM INTERFACE DESIGN

Im folgenden Kapitel werden Methoden vorgestellt, mit denen der Operateursarbeitsplatz modelliert, geprüft und weiterentwickelt wurde. Das Project ATEO ist interdisziplinär geprägt. Der Operateursarbeitsplatz wurde gemeinsam von Psychologen und Informatikern erstellt. Um diese Kooperation zu erleichtern, war es notwendig, ein gemeinsames Verständnis über die zu bewältigenden Aufgaben zu schaffen. Im ersten Schritt wurde die Hierarchical Task Analysis (HTA) eingesetzt. Diese Methode wird dazu benutzt, um ein System anhand von Aktivitäten und Zielen zu beschreiben. Das Ergebnis dieser HTA schaffte die Voraussetzung für ein gemeinsames Verständnis der Anforderungen. Darüber hinaus bildete sie die Grundlage für die Umsetzung des Operateursarbeitsplatz, sowohl für den Prototypen von Hermann Schwarz, als auch für die im Rahmen dieser Diplomarbeit erstellten Version des Operateursarbeitsplatzes. Nach der Erstellung der ersten Version mit vollem Funktionsumfang, wurde eine Heuristische Evaluation durchgeführt. Dabei überprüften Usabilty Experten den Operateursarbeitsplatz auf potentielle Usability Probleme. Da die Usability Experten keine Nutzer sind, fehlt ihnen das Wissen über den Kontext der geprüften Komponente. Die Beanstandungen basieren daher auf Regeln und nicht auf Erfahrungen, wie die von typischen Nutzern des Systems. Um diese Lücke zu schließen, wurden zusätzlich Usability Tests durchgeführt. Die Tester in diesem Verfahren waren echte User (geschulte Operateure aus dem Projekt ATEO), die echte Aufgaben durchführen. Die Durchführung wurde durch den Testleiter, Fragebogen und Logfiles dokumentiert. Die so gefundenen Defizite wurden daraufhin mittels Prototyping behoben. Die Heuristische Evaluation und die Usability Tests wurden in Iterationen durchgeführt. Die Ergebnisse flossen fortlaufend als Anpassungen in den Entwicklungsprozess ein. In dieser Phase des Projektes fand der Übergang vom Rapid Prototyping hin zum Evolutionary Prototyping statt.

3.3.1 *Hierarchical Task Analysis*

Basis für Funktionalitäten des Operateursarbeitsplatz war eine Hierarchical Task Analysis. Diese wurde von Hermann Schwarz im Zuge seiner Diplomarbeit durchgeführt. Die Analyse bildete eine der Grundlagen für die Erstellung des Prototyps. Zwei der Ziele waren es, ein besseres User Interface Design zu ermöglichen

und gleichzeitig sicherzustellen, dass alle vorher formulierten Aufgaben berücksichtigt werden. Mit Hilfe einer HTA können komplexe Systeme analysiert werden. Dazu wird eine Aufgabe oder Aktivität iterativ in ihre Teile zerlegt. Für diese Methode existiert kein scharfes Kriterium, welches anzeigt, wann die Zerlegung abgeschlossen ist. Daher liegt eine der Schwierigkeiten einer HTA darin zu bestimmen, wann diese beendet ist. Stanton [21] schlägt als Abbruchkriterium eine laufende Prüfung der Zerlegung hinsichtlich Fehlerwahrscheinlichkeit und Fehlerkosten vor. Solange das Produkt beider über einer gewählten Schwelle liegt, wird die HTA fortgeführt. Eine andere Variante ist, die HTA zu beenden, wenn allen Beteiligten die modellierte Aufgabe ausreichend verständlich ist. Beide Varianten haben den Nachteil, dass sie in ihrer Anwendung nicht ohne menschlichen Einfluss oder Intuition („Bauchgefühl“) auskommen. So lassen sich zwei verschiedene Entwürfe nicht objektiv nach Güte bewerten. Beide Varianten erfordern sorgfältige Überprüfung und möglichst viel Erfahrung. Bei der Durchführung einer HTA entsteht eine Baumstruktur. Jede Iteration, die zu einer Konkretisierung oder Verfeinerung der einer Aufgabe führt, mündet in einem neuen Blattknoten im Baum. Für jede in einem Blatt beschriebene Aufgabe wird ein eigener Plan erstellt, welcher die Ausführung dieser Aktivität beschreibt.

3.3.2 *Heuristische Evaluation*

Bei der Entwicklung des Operateursarbeitsplatzes wurde bereits früh eine Heuristische Evaluation durchgeführt. Damit konnten wertvolle Erkenntnisse für die weitere Entwicklung gewonnen werden. Im Rahmen einer Heuristischen Evaluation untersuchen Usability Experten ein ganzes System oder eine Komponente davon unter Zuhilfenahme von Heuristiken. Heuristiken sind allgemeine Regeln oder Prinzipien und somit keine konkreten Richtlinien. Die Usability Experten führen ein Analyse durch, um zu untersuchen, ob das System oder die Komponente eine Reihe von Heuristiken verletzt. Nielsen [16] führt folgende Heuristiken an:

1. Sichtbarkeit des Systemstatus
   Das System sollte den Nutzer jederzeit, in angemessener Form und Zeit über das System und seine Vorgänge informieren.
2. Übereinstimmung zwischen System und „echter“ Welt
   Das System sollte:
   - in der Sprache des Nutzers verfasst sein

- anstelle von systemorientierten Begriffen sollten Wörter, Phrasen und Konzepte zum Einsatz kommen, mit denen der Nutzer vertraut ist.
- Konventionen aus der echten Welt folgen, damit Informationen in einer für den Nutzer natürlichen und logischen Anordnung stehen

3. Nutzerkontrolle und Freiheit
   Das System sollte:
   - im Falle einer fälschlicherweise ausgewählten Funktion einen klar markierten „Notausgang“ bereithalten, der es erlaubt, diesen nicht gewünschten Zustand ohne großen Aufwand (z.B. Dialog) zu verlassen
   - eine „redo“ and „undo“ Funktion unterstützen

4. Konsistenz und Standard
   Nutzer sollten sich nicht fragen müssen, ob unterschiedliche Worte, Situationen oder Aktionen das Gleiche bedeuten. Plattformkonventionen sollten befolgt werden.

5. Fehlervermeidung
   Ungleich besser als eine gute Fehlernachricht ist sorgfältiges Design, welches das Auftreten von Problemen vermeidet. Fehleranfällige Bedingungen sollten vermieden werden oder der Nutzer sollte die Möglichkeit erhalten, eine Aktion zu bestätigen bevor sie ausgeführt wird.

6. Erkennen anstatt Erinnern
   Minimiere die Erinnerungslast des Nutzers in dem Objekte, Aktionen und Optionen sichtbar gemacht werden. Der Nutzer sollte sich keine Informationen von vorhergehenden Teilen des Dialoges merken müssen. Anweisungen zur Benutzung des Systems sollten bei Bedarf sichtbar oder einfach zugänglich sein.

7. Flexibilität und Nutzungseffizienz
   Abkürzungen können oft - unbemerkt vom ungeübten Nutzer - die Interaktion für den geübten Nutzer (Experte) beschleunigen. Damit kann ein System sowohl auf die Bedürfnisse eines Anfängers als auch auf die eines Experten eingehen. Häufige Aufgaben sollten durch den Nutzer anpassbar sein.

8. Ästethik und minimalistisches Desgin
   Dialoge sollten keine Informationen enthalten, die irrelevant sind oder selten benötigt werden. Jede zusätzliche Information in einem Dialog steht in Konkurrenz mit relevanten Informationen und verringert deren relative Sichtbarkeit.

9. Unterstütze den Nutzer beim Erkennen, Diagnose und Fehlerbehebung
   Fehlermeldungen sollten in einer klaren Weise (keine Abkürzungen oder Codes) verfasst sein. Sie sollten deutlich auf das ursächliche Problem hinweisen und möglichst auf eine Lösung verweisen.

10. Hilfe und Dokumentation
    Im besten Fall ist für die Benutzung eines Systems weder Hilfestellung noch Dokumentation erforderlich. Dennoch kann es notwendig werden, beides bereitzustellen. Diese Informationen sollten einfach zu durchsuchen sein. Sie sollten sich auf die Aufgabe des Nutzers konzentrieren und konkrete auszuführende Schritte auflisten. Zusätzlich sollten sie auch nicht zu gross sein.

Die hier angewendete Heuristische Evaluation weist einige Besonderheiten auf. Die Evaluatoren waren keine Domänenexperten, d.h., sie verfügten über keine Erfahrung auf dem Gebiet der Prozessteuerung. Daher erfolgte die Untersuchung mit Hilfe acht konkreter Aufgaben. Diese wurden zuvor im Rahmen der HTA ermittelt. Mit Hilfe dieser Vorgehensweise wurde den Evaluatoren ein Eindruck über die Funktionsweise des Operateursarbeitsplatz vermittelt. Dass der Operateursarbeitsplatz das erste System dieser Art im ATEO Umfeld war, stellte eine zusätzliche Herausforderung dar. Im vorliegenden Fall wurde die Heuristische Evaluation in zwei Phasen durchgeführt. In der ersten Phase untersuchten drei Experten die erste weitestgehend funktionsfähige Version des Operateursarbeitsplatz. Dieser Version fehlte noch die Netzwerkanbindung an die Trackingkomponente SAM. Im Normalfall ist der Operateursarbeitsplatz mit SAM über ein Netzwerk verbunden und erhält Inputdaten von SAM. Für die Streckenvorschau auf dem Operateursarbeitsplatz wurden Inputdaten zufällig im Operateursarbeitsplatz erzeugt, um ein normales Verhalten zu simulieren. Nach dieser ersten Heuristischen Evaluation wurden die dort gesammelten Ergebnisse ausgewertet. Die daraus abgeleiteten Anpassungen am Operateursarbeitsplatz wurden im Zuge einer weiteren Iteration des Prototyping Entwicklungsprozesses eingarbeitet. In den Abbildungen 3 bis 5 auf Seite 29 ist am Beispiel der Auditiven Hinweise die fortschreitende Entwicklung dargestellt und im folgenden Abschnitt 3.3.2.1 erläutert. In der zweiten Phase analysierten drei weitere Experten den weiterentwickelten Operateursarbeitsplatz. Die Evaluation wurde wieder mit Hilfe der gleichen acht Aufgaben aus der ersten Runde durchgeführt. Die Ergebnisse sind auch hier in einer weiteren Iteration des Softwareentwicklungsprozesses umgesetzt worden.

3.3.2.1 *Entwicklung über die Prototyping Iterationen – ein Beispiel*

Die Aufgabe des Elementes und die Details der Umsetzung können dem Abschnitt 4.3.2.5 ab Seite 59 entnommen werden. Hier sollen nur die Veränderungen über die Prototyping Iterationen und deren Gründe am Beipiel des Elementes Auditive Hinweise erläutert werden.

In Abbildung 3 ist das Element Auditive Hinweise der Version 1.1.7 zu sehen. Um einen Hinweis zu geben müssen zwei Schritte durchgeführt werden. Als erstes muss der Adressat ausgewählt werden, danach der Hinweis. In dieser Version war eine Bedienreihenfolge nicht klar zu erkennen, alle Buttons konnten unmittelbar geklickt werden. Das führte häufig zu Fehlbedienungen, da oftmals nur ein Hinweis aber kein Adressat ausgewählt wurde.

Als Lösung für dieses Problem wurden in Version 2.2.0 die Hinweisbuttons im Initialzustand des Elementes gesperrt. Über die Hinweisbuttons wurde ein transparenter grauer Morph gelegt. Somit waren die Buttons für den Nutzer zwar noch ausgegraut sichtbar, aber nicht mehr verfügbar. In Abbildung 4 ist der initiale Zustand dargestellt. Erst nachdem der Operateur einen Adressaten ausgewählt hatte, konnten die Hinweisbuttons angeklickt werden. Trotzdem traten Fehlbedienungen weiterhin auf, da die Hinweisbuttons nicht als gesperrt wahrgenommen wurden.

In Abbildung 5 ist die bisher letzte Version 2.8.4 dargestellt. Die Adressatenleiste verlor ihren festen Platz im Element und wurde immer zentriert über dem jeweils gewählten Hinweis nach dem Klick eingeblendet. Nachdem in der Adressleiste ein Adressat gewählt wurde, wurde sie wieder ausgeblendet. Die Unsicherheit hinsichtlich der Reihenfolge aus den alten Versionen wurde ausgeräumt und die Bedienung des Elementes wurde intuitiver. Der zusätzlich gewonnene Platz konnte für einen weiteren Hinweis genutzt werden. In dieser Version des Elementes reduzierten sich die Fehlbedienungen auf ein akzeptables Maß.

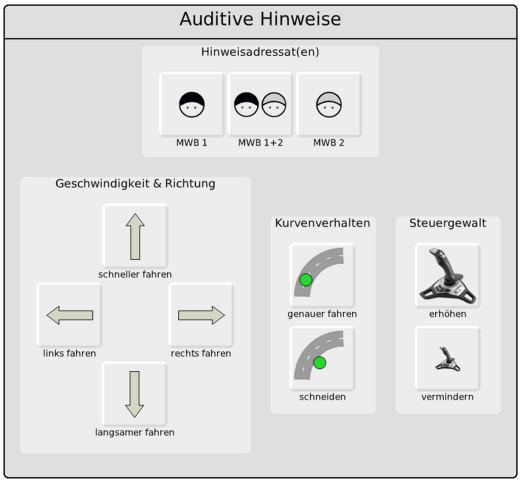

Abbildung 3: Element Auditive Hinweise aus der Version 1.1.7

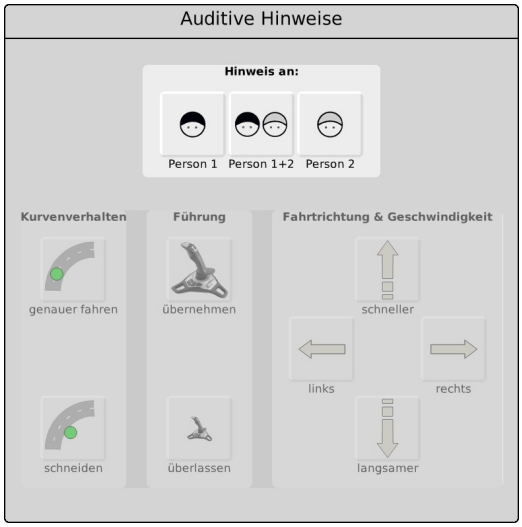

Abbildung 4: Element Auditive Hinweise aus der Version 2.2.0

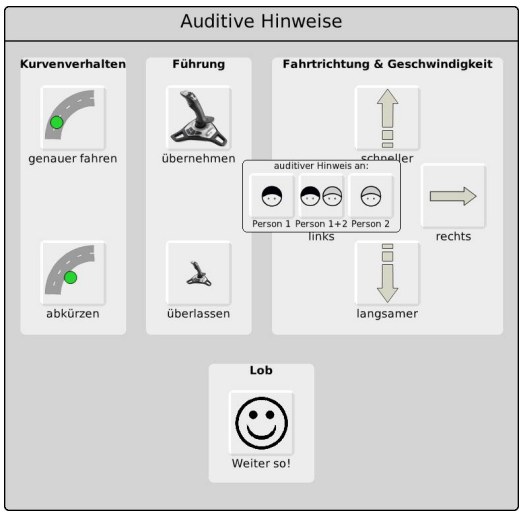

Abbildung 5: Element Auditive Hinweise aus der Version 2.8.4

3.3.3 *Usability Test*

Die Durchführung der bisher vorgestellten Methoden dienten als Vorbereitung für den Usability Test. Dieser wird in der Literatur manchmal auch als User Test bezeichnet. In [14] wird erläutert, warum die Bezeichnung Usability Test zu bevorzugen ist. Der User ist in dem Verfahren eher die Quelle der Information und weniger der Untersuchungsgegenstand. Usability hingegen ist der eigentliche Untersuchensgegenstand. Daher wird im Folgenden die Bezeichnung Usability Test verwendet. Das Konzept Usability beschreibt die Interaktion zwischen einem Nutzer und einem System. Sie wird in der „EN ISO 9241-11“ definiert als „...das Ausmaß, in dem ein Produkt, System oder ein Dienst durch bestimmte Benutzer in einem bestimmten Anwendungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.“

Die Güte von Usability kann nach „EN ISO 9241-11“ anhand von drei Kriterien beurteilt werden:

- Effektivität: Die Genauigkeit und Vollständigkeit, mit der Benutzer ein bestimmtes Ziel erreichen.
- Effizienz: Der im Verhältnis zur Genauigkeit und Vollständigkeit eingesetzte Aufwand, mit dem Benutzer ein bestimmtes Ziel erreichen.
- Zufriedenstellung: Freiheit von Beeinträchtigungen und positive Einstellung gegenüber der Nutzung des Produkts.

Dadurch bedingt wird beim Usability Testing eine Anzahl abhängiger Variablen untersucht. Beispielsweise sind das die Geschwindigkeit der Aufgabenbewältigung, die Anzahl der Fehler bei der Bearbeitung der Aufgabe oder das Maß subjektiver Zufriedenheit. Komplexere Variablen sind beispielsweise die mentale Belastung oder das Situationsbewusstsein. Im Falle des Operateursarbeitsplatzes wurden die Variablen durch Logfileauswertungen und Befragungen erhoben. Die folgenden Eigenschaften eines Usability Tests sind die bedeutsamsten:

- Das vorrangige Ziel ist die Verbesserung der Usability des Untersuchungsgegenstandes – in diesem Fall der Operateursarbeitsplatz.
- Teilnehmer der Untersuchung sind echte Nutzer, in diesem Fall geschulte Operateure aus dem ATEO Projekt.
- Die Teilnehmer bearbeiten echte Aufgaben, in diesem Fall die Überwachung der Trackingsimulation SAM.

- Das Verhalten und die Äußerungen der Teilnehmer werden beobachtet, in diesem Fall via Logfile, Versuchsleiter und Fragebogen.
- Die Daten des Tests werden analysiert, Probleme im Design identifiziert und daraus Empfehlung für Verbesserungen abgeleitet.

Usability Tests sollten möglichst mit Design Guidelines und Heuristischer Evaluation kombiniert werden. Vor allem die Heuristische Evaluation und der Usability Test ergänzen sich hervorragend. Das gleiche Interface wird aus zwei verschiedenen Perspektiven validiert. Die Usability Evaluatoren betrachten das Interface aus einer externen und allgemeinen Perspektive. Die Nutzer nehmen im Gegensatz dazu eine konkrete und partizipierende Perspektive ein. Damit wird das Interface aus verschiedenen Perspektiven überschneidend validiert. Da es sich bei dem Usability Test um eine echte Anwendungssituation mit all ihrer Komplexität handelt, ist die Durchführung sehr herausfordernd. Trotzdem sind auch umfangreiche Tests keine Garantie dafür, dass jeder kritische Designfehler gefunden wird. Die Ergebnisse sind auch von den Versuchsleitern, den verwendeten Tests und den Methoden abhängig. Ein Usability Test erfordert ein hohes Maß an Anpassung an die vorliegenden Rahmenbedingungen. Wird er strikt nach seinen allgemeinen Regeln durchgeführt, ist das Verfahren bestenfalls uneffektiv. Die Entscheidung, welche Variablen untersucht werden, ist abhängig von dem Bereich in dem die Untersuchung stattfindet. Im Bereich Prozessführung sind Maße für Leistung, Trackinggeschwindigkeit, Genauigkeit und subjektive Zufriedenheit nicht ausreichend. Daher wurden zusätzlich Situationsbewusstsein und Mentale Beanspruchung untersucht.

SITUATIONSBEWUSSTSEIN Situationsbewusstsein nach Endsley [6] ist das „...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.“ Das Situationsbewusstsein ist ein Größe, welche immer maximal sein sollte. Für die Messung wurde das Operateursarbeitsplatzelement „Messung Situationsbewusstsein“ verwendet. Die Funktionsweise wird in Abschnitt 4.3.4.3 erläutert.

MENTALE BEANSPRUCHUNG Der Begriff mentale Beanspruchung beschreibt die Intensität der Informationsverarbeitung. Sie wird durch Faktoren wie Schwierigkeit oder Komplexität einer Aufgabe beeinflusst. Im Gegensatz zum vorher beschriebenen Situationsbewusstsein ist das ideale Maß von mentaler Beanspru-

chung Gegenstand intensiver Diskussionen. Zu niedrige oder zu hohe mentale Beanspruchung mindern die Leistung eines Operateurs. Die Herausforderung beim Design eines Prozessführungssystems ist die Balance, um zu vermeiden, dass die Benutzung nicht in 99% Langeweile und 1% reinem Terror [22] münden. Zur Messung der subjektiven mentalen Beanspruchung wurde eine Skala verwendet, mit deren Hilfe der Operateur eine Selbsteinschätzung vornehmen konnte. Zusätzlich wurden die Reaktionszeiten bei der Messung des Situationsbewusstseins herangezogen.

TRACKINGLEISTUNG Die Güte der Trackingleistung der MWB wurde ebenfalls untersucht. Die Optimierung von Geschwindigkeit und Genauigkeit des Trackings war die Hauptaufgabe des Operateurs. Beispielsweise wurden dazu die Logfiles auf Geschwindigkeit, Anzahl und Genauigkeit der harten und weichen Eingriffe des Operateurs analysiert.

USABILITY RATING Usability Rating ist der subjektive Eindruck des Operateurs. Nach einem Versuch wurde von dem Operateur ein Fragebogen ausgefüllt. Mit dessen Hilfe wurde das Usability Rating gemessen.

Insgesamt wurden 78 Usability Tests durchgeführt mit je einem Operateur und zwei MWB. Insgesamt nahmen also 78 Operateure und 156 MWB teil. Damit ist die Stichprobe hinreichend groß, um eine ausreichende statistische Aussagkraft sicherzustellen. Die Zwischenergebnisse der Usability Tests wurden zwischen den einzelnen Untersuchungen eingarbeitet und waren damit Teil der darauffolgenden Usability Tests.

4 IMPLEMENTIERUNG

4.1 ANALYSE DES ALTSYSTEMS

Im Rahmen seiner Diplomarbeit erstellte Hermann Schwarz die erste Softwareversion des Operateursarbeitsplatzes in Squeak. Dem ging eine detaillierte Analyse der Aufgaben eines Operateurs im Kontext der ATEO Versuchsanordnung voraus. Dazu wurden die Verfahren HTA und Objektorientierte Analyse (OOA) eingesetzt. Der darauf folgende Entwurf und die Implementierung basierten auf diesen Ergebnissen.

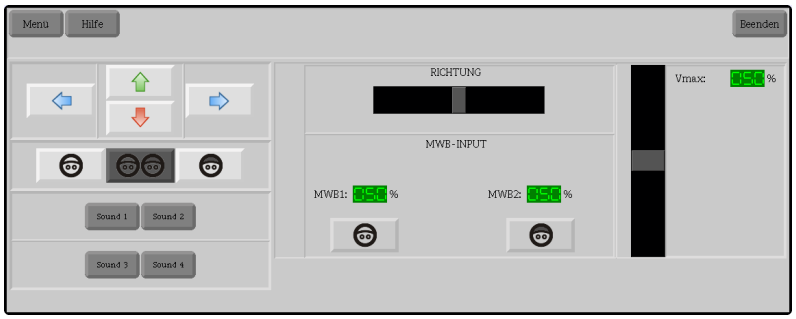

Abbildung 6: Operateursarbeitsplatz von Hermann Schwarz (Version 0.1)

Die einzelnen Elemente sind hier in einem Fenster zusammengefasst. Um mit verschiedenen Zusammensetzungen der einzelnen Elemente Ausbaustufen bilden zu können, muss diese Komposition aufgebrochen und jedes Element in einem eigenen Fenster eingebettet werden.

4.1.1 *Element: MWB-Input*

Dieses GUI-Element ist im Rahmen der vorhergehenden Studienarbeit überarbeitet worden. In der Version von Hermann Schwarz änderten die Buttons mit jedem Klick ihr Grösse, um dem Operateur Rückmeldung über die eingestellte Verteilung zu geben. Im Zuge der Überarbeitung wurde dieses Verhalten geändert, so daß die Grösse der Buttons konstant bleibt. Rückmeldung geben stattdessen die in ihnen enthaltenen Grafiken. Diese ändern ihre Grösse entsprechend der eingestellten Verteilung. Ein kleiner Anteil des entsprechenden MWB resultiert in einer kleineren Darstellung der dazugehörigen Grafik.

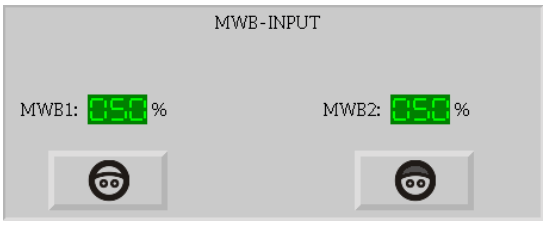

Abbildung 7: Element Steuerungsanteil, Verteilung 50/50 (Version 0.1)

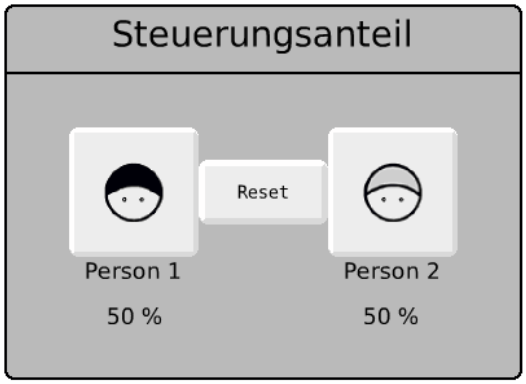

Abbildung 8: Element Steuerungsanteil, Verteilung 50/50 (Version 2.8.4)

Auch die Höhe der Wertänderung je Klick wurde verändert. Im Prototypen löste ein Klick eine Umverteilung des Steuerungsanteils von 25% aus. Dieser Wert wurde auf 5% gesenkt. Das ermöglicht eine feinere Abstimmung durch den Operateur.

Das Aussehen wurde ebenfall überarbeitet. Die Prozentangaben sind unter die beiden Buttons verschoben worden. Für die Grafiken zur Darstellung der MWB wurden neue Versionen eingesetzt.

Zusätzlich wurde ein neuer Button zwischen die beiden bestehenden Buttons hinzugefügt. Dieser Reset Button erlaubt es, mit einem Klick in die Ausgangsverteilung von 50% je MWB zurückzukehren. Die Überarbeitung und Details der neuen Implementierung sind in meiner vorhergehenden Studienarbeit [12] näher beschrieben.

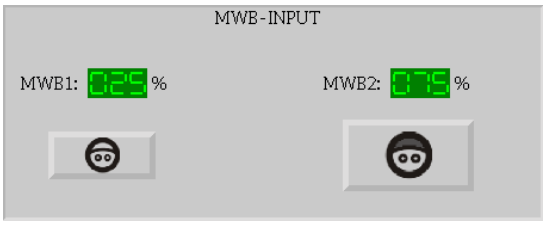

Abbildung 9: Element Steuerungsanteil, Verteilung 25/75 (Version 0.1)

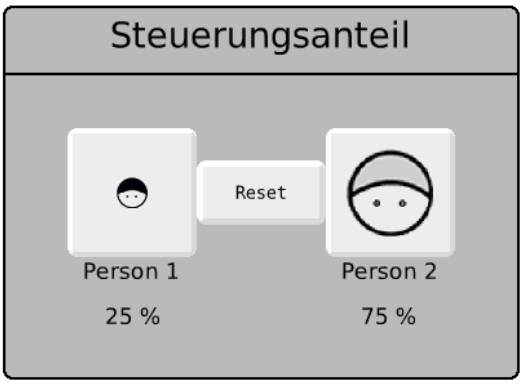

Abbildung 10: Element Steuerungsanteil, Verteilung 25/75 (Version 2.8.4)

4.1.2 *Element: Vmax*

Auch das Aussehen und die Funktionnalität dieses Elementes wurde stark überarbeitet. Zur besseren Einordnung durch den Operateur wurde der Slider um eine Skala erweitert. Der aktuell eingestellte Wert des Geschwindigkeitslimits wird immer entsprechend der Positionen des Sliders platziert. Das soll die Interpretation des Elementes durch den Operateur erleichtern. Der Reset Button ermöglicht eine Rückkehr zum Ausgangswert des Geschwindigkeitslimits – analog zu den restlichen Reset Buttons des Operateursarbeitsplatzes.

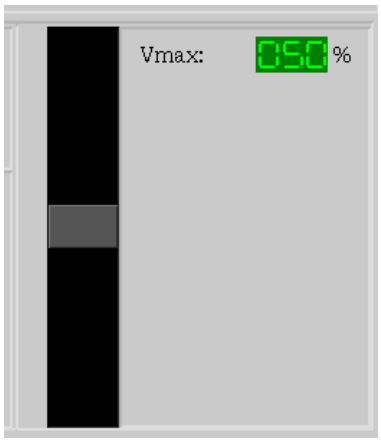

Abbildung 11: Element Vmax (Version 0.1)

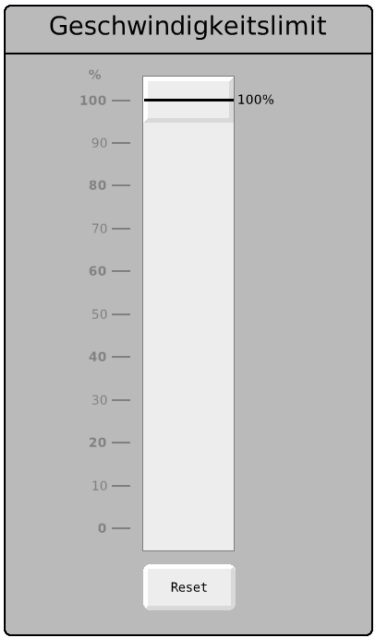

Abbildung 12: Element Geschwindigkeitslimit (Version 2.8.4)

4.1.3 *Element: Richtung*

Durch die fortlaufende fachliche Überarbeitungen wurde auch dieses Element grundlegend geändert. Mit Hilfe dieses Elementes kann der Oparateur die horizontalen Bewegungen des Fahrobjektes in der eingestellten Richtung unterbinden. Der Schieberegler wurde durch zwei Buttons ersetzt und ein Reset Button eingefügt.

Abbildung 13: Element Richtung (Version 0.1)

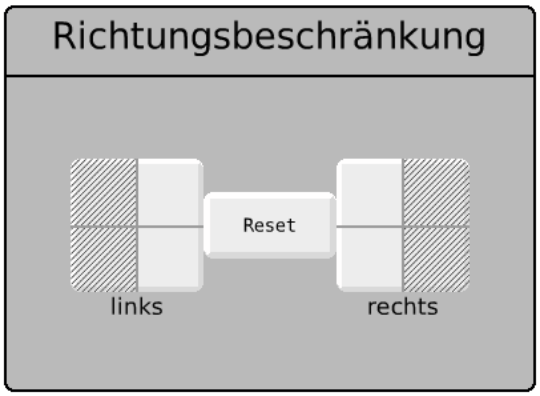

Abbildung 14: Element Richtungsbeschränkung (Version 2.8.4)

4.1.4 *Element: Visuelle Hinweise*

In der Version von Hermann Schwarz hat der Oparateur die Möglichkeit, den MWB während der Fahrt visuelle Hinweise in Form von auf der Strecke eingeblendeten Pictogrammen zu geben. Das ist in dieser Version jedoch auf horzontale und vertikale Auslenkung beschränkt. Die überarbeitete Version umfasst die bestehende Funktionalität und ermöglicht darüber hinaus noch Hinweise auf Gabelungen und Hindernisse.

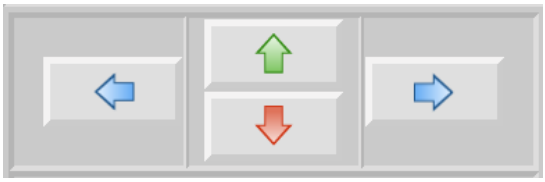

Abbildung 15: Element Visuelle Hinweise (Version 0.1)

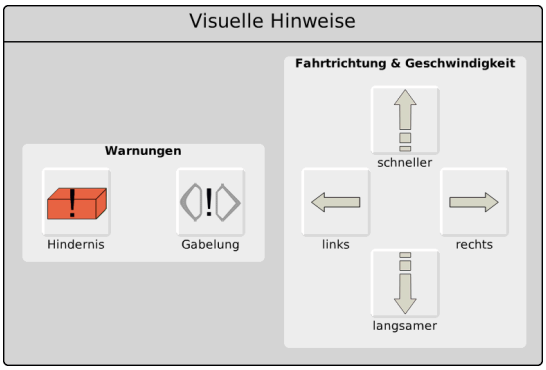

Abbildung 16: Element Visuelle Hinweise (Version 2.8.4)

4.1.5 *Element: Auditive Hinweise*

Auch hier sind deutliche Änderungen zum aktuellen Entwurf erkennbar. Vor allem der Unterschied in der Bezeichnung der Buttons ist auffällig. In der Version von Hermann Schwarz wurden die Buttons für die Sounds mit generischen Namen versehen. Damit fehlte dem Operateur eine Hilfestellung, um zu erkennen hinter welchem Button sich welcher auditiver Hinweis verbirgt. Diese Information musste somit entweder gemerkt und erfolgreich abgerufen oder neu beschafft werden. In der Version 2.8.4 sind die Buttons mit Grafiken versehen. Diese sollen zusammen mit der genaueren Beschriftung der Buttons die jeweiligen Funktion der auditiven Hinweise erklären. Auch die Anzahl der auditiven Hinweise hat sich von initial vier auf inzwischen neun erhöht.

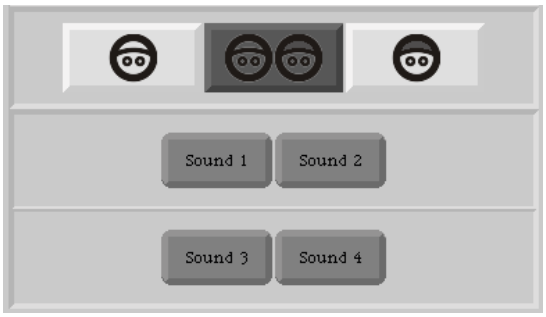

Abbildung 17: Element Auditive Hinweise (Version 0.1)

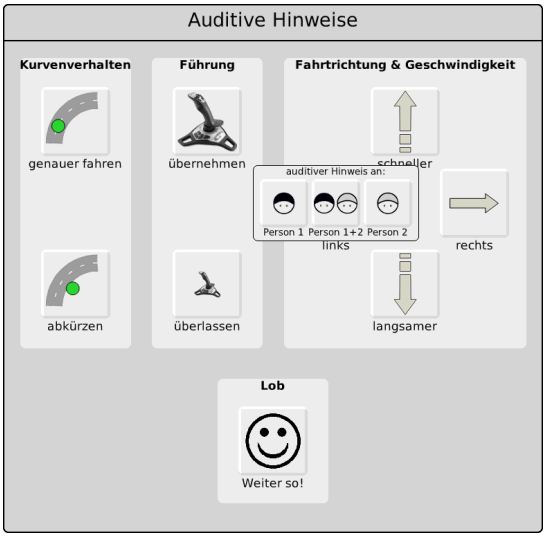

Abbildung 18: Element Auditive Hinweise mit aktivierter Adressatenauswahl (Version 2.8.4)

Es wurde geändert, wie Adressaten für auditive Hinweise ausgewählt werden. In der ersten Version wurde zunächst der Adressat ausgewählt. Danach wurden die Buttons mit den auditiven Hinweis zur Auswahl freigeschaltet. Diese Reihenfolge wurde in der aktuellen Fassung umgekehrt. Erst wird der auditive Hinweis gewählt, danach öffnet sich das Auswahlfenster für die Adressaten.

4.1.6 *Element: Joystickauslenkungen*

Dieses Element ist ebenfalls in der vorangegangenen Studienarbeit überarbeitet worden. Vor allem die in der ersten Version vorgenommene Vereinfachung der dargestellten Auslenkungen wurde in eine genauere Abbildung überführt. Zusätzlich werden in der Anzeige die harten Eingriffe zurückgemeldet. Detailliertere Erläuterungen können der Studienarbeit [12] entnommen werden.

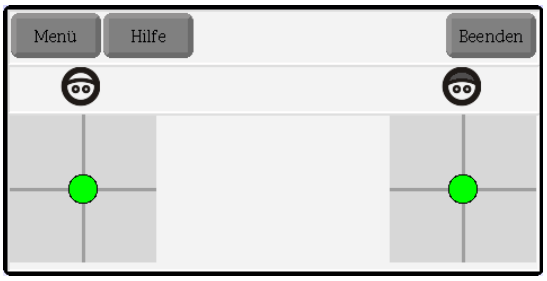

Abbildung 19: Element Joystickauslenkungen (Version 0.1)

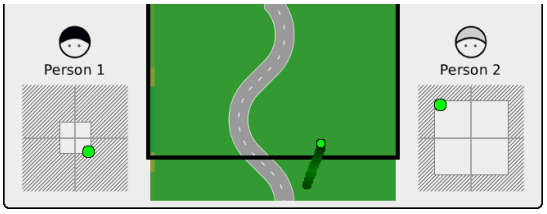

Abbildung 20: Elemente Joystickauslenkungen (Version 2.8.4)

4.2 SOFTWAREARCHITEKTUR DES OPERATEURSARBEITSPLATZES

Der Begriff Softwarearchitektur beschreibt die Struktur eines Softwaresystems, die Beziehungen, die Interaktionen seiner Bestandteile und die räumliche Verteilung. Bekannte Architekturen sind u.a. die Schichtenarchitektur oder die Model-View-Controller-Architektur (MVC-Architektur). Der Operateursarbeitsplatz orientierte sich an der MVC-Architektur. Diese ist eine objektbasierte Architektur. Die Grundidee von MVC ist die Aufteilung des Systems in folgende Teile:

- Model: Dieser Teil enthält die Daten
- View: Dieser Teil übernimmt die Darstellung der Daten und nimmt Benutzereingaben entgegen
- Controller: Dieser Teil steuert eine oder mehrere View-Komponenten und verarbeitet die von ihnen übergebenen Benutzereingaben

Im Folgenden wird erläutert, warum die MVC-Architektur für den Operateursarbeitsplatz im ATEO Kontext eingesetzt wurde.

Lose Kopplung Eine Anforderung an den Operateursarbeitsplatz war es, verschiedene Ausbaustufen erstellen zu können. Eine Ausbaustufe ist eine Zusammenstellung aus den zur Verfügung stehenden Einzelelementen des Operateursarbeitsplatzes. Zwei Ausbaustufen unterscheiden sich durch ihre Komposition der Einzelelemente. Das bedeutet, dass nicht jedes Einzelelement

für jedes Experiment genutzt wird. Gleichzeitig teilen sich verschiedene Elemente die gleichen Daten. Würden also Daten in einem Einzelelement des Operateursarbeitsplatzes gespeichert, stünde diese Information in einer Ausbaustufe ohne dieses Einzelelement nicht zur Verfügung. In diesem Fall könnten andere Elemente ihre Aufgabe nur teilweise oder gar nicht erfüllen. Dadurch ist es sinnvoll, das Datenmodell vom restlichen System zu trennen.

Wartbarkeit Durch die Architektur bedingt sind die Einzelelemente weitgehend getrennt. Damit beschränken sich die Auswirkungen bei Anpassungen oder Fehlerbehebungen auf einen überschaubaren Bereich. Das sollte im Normalfall nicht über die betroffene Komponente hinausgehen.

Erweiterbarkeit Änderungen, z.B. für neue Elemente des Operateursarbeitsplatzes, sollten ohne größeren Aufwand realisierbar sein. Das wird durch die MVC-Architektur ermöglicht.

Arbeitsteilung Durch die klare Trennung der Teile kann an verschiedenen Stellen gleichzeitig am Projekt gearbeitet werden. Auch die Abgrenzung der Arbeitspakete wird dadurch vereinfacht. Für künftige Arbeiten stellt das einen großen Vorteil dar.

Synchronisierte views Durch die MVC-Architektur sind model und view von einander getrennt. Jeder view holt sich die zur Darstellung notwendigen Daten vom model. Damit ist sichergestellt, dass alle auf dem gleichen Datenbestand arbeiten. Somit wird im Hinblick auf die Daten Synchronität erreicht.

Bei der klassischen Variante der MVC-Architektur ist jedem view ein controller zugeordnet. In der vorliegenden Implementierung gilt das nicht für jeden view. Hier teilen sich mehrere views einen controller. Dieser ist in der Klasse GuiMasterControl untergebracht und ist seinerseits gleichzeitig der controller für diese Klasse. Die Klassen ohne eigenen Controller können nicht durch den Operateur manipuliert werden. Sie sind reine Anzeigeelemente wie beispielsweise die Klassen für die Elemente für den Systemstatus oder die Anstrengung.

4.2.1 *Komponente model im Operateursarbeitsplatz*

Die Komponente model ist in der MVC-Architektur für die Datenhaltung zuständig. In der Literatur wird hier manchmal auch die Geschäftslogik angesiedelt. Im Oparateursarbeitsplatz enthält model neben den Daten auch einen kleinen Teil der Logik. Die Komponente model im Oparateursarbeitsplatz wird durch drei

Klassen ausgefüllt. Diese Klassen sind fast ausschließlich Datenspeicher und werden von den anderen Klassen direkt zur Ablage und Abfrage von Daten genutzt. Die mehrfache Verwendung einzelner Daten in verschiedenen Views wird dadurch erleichtert.

4.2.1.1 *Klasse modelData*

Die Einordnung der model-Klassen in die Architektur des Operateursarbeitsplatzes ist im Anhang unter A.5 auf Seite 96 dargestellt.

Diese Klasse ist ein reiner Datenspeicher. Die Methoden realisieren bis auf zwei Ausnahmen ausschliesslich Lese- oder Schreibzugriffe. Die beiden anderen Methoden initialisieren mit Standardwerten die Daten oder setzen sie z.B. nach einer gefahrenen Strecke wieder zurück. Konzeptionell werden hier die Daten gespeichert, welche für den Betrieb des Operateursarbeitsplatzes benötigt werden. Für die Daten die der Oparateursarbeitsplatz über das Netzwerk sendet oder empfängt, sind die im Folgenden vorgestellten Klassen modelImport und modelExport zuständig.

4.2.1.2 *Klasse modelImport*

Neben den Daten enthält diese Klasse auch Verarbeitungslogik. Sämtliche an den Oparateursarbeitsplatz gesendete Daten werden hier gespeichert. Diese können aus verschiedenen Quellen stammen. Das sind z.B. Daten von SAM, wie Geschwindigkeit oder Auslenkungen der MWB, oder das Anstrengungsmaß für jeden MWB von einer der beiden MW-Komponenten. Für die Kommunikation über das Netzwerk wird ein eigenes Format eingesetzt. Dazu werden die einzelnen Daten in eine feste Reihenfolge gebracht und mit einem festgelegten Separator voneinander getrennt. Die auf diese Art erstellten Informationspakete werden Frames genannt. Eine genauere Beschreibung der Frames ist in der Studienarbeit [12] nachzulesen. Die Klasse verfügt über Methoden, um die Frames wieder in seine Bestandteile zu zerlegen. Dazu werden sie davor als String dem Socket entnommen. Um Fehler bei der weiteren Verarbeitung zu vermeiden, gibt es darüber hinaus noch weitere Konvertierungsmethoden. Mit deren Hilfe wird sichergestellt, dass im Datenspeicher nur Inhalte mit dem erwarteten Format gespeichert werden. Beispielsweise soll als Wert für den Steuerungsanteil nur eine Zahl oder für eine Joystickauslenkung nur ein Punkt gespeichert werden. Daten mit dem falschen Format werden verworfen. Der Zugriff der restlichen Komponenten des Operateursarbeitsplatzes auf die gespeicherten Daten erfolgt ausschließlich durch Lesemethoden. Überschrieben werden die entsprechenden Daten nur von den Methoden, die die Frames verarbeiten.

4.2.1.3 *Klasse modelExport*

Die Aufgabe dieser Klasse ist das Bereitstellen von Daten aus dem Operateursarbeitsplatz. Für die Daten existieren nur Schreib-

methoden. Methoden zum Lesen sind nicht erforderlich. Auch hier ist ein Teil der Methoden für die Verarbeitung verantwortlich. Neben einer Methode zum Zurücksetzen der Inhalte ist eine andere für die Erstellung der Netzwerkframes verantwortlich. Weitere Details zu den Netzwerkframes können der Studienarbeit [12] entnommen werden.

4.2.2 *Komponente view im Operateursarbeitsplatz*

Dieser Teil der Architektur ist verantwortlich für die Darstellung der Benutzeroberfläche und für das Entgegennehmen von Benutzerinteraktionen. Ein view kennt den Teil des models den es repräsentiert sowie seinen controller. Ein Teilelement des Operateursarbeitsplatzes, wie z.B. die Streckenanzeige oder der Systemstatus, entspricht einer Klasse. Diese GUI Klassen zusammengenommen bilden den view für den Operateursarbeitsplatz.

4.2.3 *Komponente controller im Operateursarbeitsplatz*

Die controller sind das Bindeglied zwischen model und view. Sie nehmen Benutzereingaben von den views entgegen und werten diese aus. Auf Basis der Eingaben werden bei Bedarf auch die Daten in model verändert. Im Operateursarbeitsplatz existert nicht zu jedem view ein eigener Controller. Das betrifft Elemente, die keine Interaktion erlauben und nur Informationen darstellen. Das sind u.a. die Elemente Systemstatus, Videobild oder Streckenvorschau. In der Klasse OPGuiMasterControl befindet sich die Hauptprogrammschleife. Dort wird an den entsprechenden Zeitpunkten oder bei Änderungen von model eine Aktualisierung der Darstellung ausgelöst. Der Systemstatus oder die Anstrengung werden nur zu Beginn einer Fahrt aktualisiert. Mit Elementen, wie z.b. Auditive Hinweise, Geschwindigkeitslimit oder Steuerungsanteil, kann der Operateur interagieren. Deren Eventhandler agieren als controller. Je Klasse kann es ein Reihe von Eventhandlern geben. In Smalltalk werden die Eventhandler als ActionSelector bezeichnet. Für diese Elemente sind einige Methoden der Klasse OPGuiMasterControl ebenfalls controller, da nicht nur Operateurseingaben eine Aktualisierung der Darstellung notwendig machen sondern auch ein Streckenwechsel. Dieser erfordert einen Reset mancher Elemente, wie z.B. Steuerungsanteil oder Joystickauslenkung. Diese müssen mit Beginn einer neuen Fahrt in ihren Ausgangszustand zurückgesetzt werden.

4.3 ENTWICKLUNG ÜBER DIE VERSIONEN

4.3.1 *Version von Hermann Schwarz*

Die Machbarkeitsstudie berücksichtige die ebenfalls erarbeiteten Styleguides und Metriken zur GUI-Entwicklung zum Überwachen komplexer Systeme. Einer der deutlichsten Unterschiede zur aktuellen Version ist die Zusammenfassung fast aller Einzelelemente in einem grossen Morph. Die Anzeige der Joystickauslenkungen ist als ein eigenes Element implementiert. Die beiden Buttons „Menü“ und „Hilfe“ wurden in den späteren Versionen nicht mehr umgesetzt. Die restlichen Elemente sind immer noch Teil des Operateursarbeitsplatzes. Auch die Grafiken für die Darstellung der MWB werden, wenn auch in leicht modifizierter Variante, in der aktuellen Version des Operateursarbeitsplatzes verwendet. Die markante digitale Darstellung der Zahlen zur Anzeige der Geschwindigkeit und des Steuerungsanteils und die verwendete Schriftart sind in den späteren Versionen überarbeitet worden. In der Version 2.8.4 sind einige Elemente hinzugekommen, wie beispielsweise Streckenvorschau, Anstrengung oder Systemstatus. Einige dieser Elemente sind erst nach Abschluss von Hermann Schwarz Arbeit überhaupt konzipiert worden. Hermann Schwarz Version hat vor allem zu Beginn der Arbeiten einen wichtigen Beitrag geleistet, da sie als Diskussionsgrundlage mit der Fachseite und als Anregung für Implementierungsalternativen diente. Auch wurde durch seine Machbarkeitsstudie gezeigt mit welchem Aufwand ein Operateursarbeitsplatz zu erstellen ist. Die Möglichkeit des Studiums des Prototypen erleichterten die Einarbeitung in das Thema Operateursarbeitsplatz und ermöglichten ein besseres Verständnis über die Zielstellung der eigenen Arbeit.

4.3.1.1 *Klasse OpWindowRectangleMorph*

Die Klasse OpWindowRectangleMorph bildet die Basis für den überwiegenden Teil der Fenster des Operateursarbeitsplatzes. Der Darstellung im Klassendiagramm sind diese Beziehungen ebenfalls zu entnehmen. Das Klassendiagramm befindet sich im Anhang unter A.5. Der Großteil der visuellen Eigenschaften und Verhaltensaspekte können hier festgelegt werden. Dadurch sind Anpassungen für Eigenschaften wie Schriftarten, Farbe der Elemente, Rahmen oder Buttons einfach und zentral zugänglich. Die Positionen der einzelnen Elemente befinden sich in einem Dictionary. Dieses wird im Rahmen der „initialize“ Methode der Klasse „OpWindowRectangleMorph“ angelegt und mit Schlüssel-Werte Paaren befüllt. Als Schlüssel werden die Klassennamen der Elemente verwendet. Die Position der Streckenvorschau findet sich also im Wert des dazugehörigen Schlüssels „OPTrackPreview“.

Damit kann das Layout des gesamten Operateursarbeitsplatzes an einem zentralen Ort angepasst werden. Die einzelnen Elemente, wie die Streckenvorschau oder das Videobild, beziehen ihre Positionsinformationen im Rahmen ihrer eigenen initialize Methode. In dieser rufen sie eine von der Oberklasse geerbte Methode „myPosition“ auf mit dem eigenen Klassennamen als Parameter und erhalten damit die vorher festgelegte Position. Weiterhin werden Methoden zur Erstellung von Bausteinen, wie zum Beispiel Buttons, zur Verfügung gestellt. Die Anordnung der einzelnen Teile innerhalb der Operateursarbeitsplatzelemente wird durch Layoutpolicies gesteuert. Durch seine Abstammung von der Klasse OpWindowRectangleMorph verfügt jedes Operateursarbeitsplatzelement bei Erstellung über ein in der Oberklasse festgelegtes Basislayout. Bei dieser Grundeinstellung werden neue Elemente horizontal und vertikal zentriert. Der Titel einer Instanz wird ebenfalls durch eine vererbte Methode gesetzt. Alle bisher implementierten oder geplanten Operateursarbeitsplatzelemente benötigen Zugriff auf die Datenhaltungsobjekte „modelData“, „modelExport“ und „modelImport“. Zu diesem Zweck werden in „OpWindowRectangleMorph“ für jede der drei vorgenannten Objekte Instanzvariablen angelegt. Die Datenhaltungsobjekte sind als Singletons implementiert. Damit erhalten alle Instanzen durch ihre geerbten Instanzvariablen Zugriff auf die gleichen Datenspeicher. Dadurch teilen sie sich die gleichen Objektinstanzen, die zum Datenaustausch genutzt werden. Ein Teil der Funktionalität des Operateursarbeitsplatzes wird dadurch erst ermöglicht. Ein Beispiel davon ist die Veränderung der Joystickanzeigen in der Streckenvorschau, wenn ein harter Eingriff erfolgt, z.B. bei Veränderung der Geschwindigkeit.

4.3.1.2 *Blindklicks*

Eine Anforderung an den Operateursarbeitsplatz ist die Aufzeichnung sämtlicher Klicks des Operateurs, die keine Aktionen auslösen. Das sind sämtliche Klicks, deren Ziel kein Button oder ein Schieberegler ist und die somit keinerlei Funktionalitäten auslösen. Diese Klicks werden im Folgenden Blindklicks genannt. Blindklicks sollen registriert und an SAM übermittelt werden. In der Komponente SAM werden die Koordinaten der Blindklicks in das Logfile geschrieben. Alle 30 ms wird ein neuer Eintrag in das Logfile vorgenommen. Damit wäre es theoretisch möglich, einzelne Blindklicks zu verlieren. Tests haben gezeigt, dass es nur sehr schwer zu schaffen ist, mehrere Klicks in derart kurzer Zeit zu produzieren. Damit ist dieser Fall sehr selten. Zusätzlich ist eine Vernachlässigung auch inhaltlich vertretbar. Aus diesen Gründen wurde auf eine feinere Erfassung verzichtet. Um Blindklicks zu registrieren, erhält die Klasse, konkret der Morph, einen Eventlistener der Klicks verarbeitet. Dadurch ist der gesamte sichtbare

und damit klickbare Bereich des Operateursarbeitsplatzes durch Eventlistener abgedeckt. Somit wird jeder Klick erfasst.

4.3.1.3 *Mauscursor*

Zur Verbesserung der Sichtbarkeit wird die Gestalt des Mauscursors für die Dauer eines Experimentes angepasst. Das wird durch eine Überladung der Methode `showTemporaryCursor` erreicht.

Abbildung 21: Standard Mauscursor

Abbildung 22: Angepasster Mauscursor

Die Datei mit dem Bild des neuen Cursors wird als Parameter übergeben. Der hier aufgezeigte Weg zur Änderung unterliegt einer Einschränkung. Wird der geänderte Cursor über einen Bereich eines Fensters bewegt, bei dem er das Aussehen ändert um eine Funktionalität zu signalisieren, beispielsweise um ein Fenster in der Größe anzupassen, wechselt er danach wieder in seine Ursprungsform zurück. Das ist für den Operateursarbeitsplatz in der aktuellen Version kein Problem, da es keine Bereiche mit derartigem Verhalten gibt. Trotzdem sollte diese Einschränkung für künftige Entwicklungen beachtet werden.

4.3.1.4 *modelData*

Die Klasse modelData dient als Container und Austauschmedium von Daten. Durch die Nutzung von modelData konnte ein direkter Datenaustauch zwischen den Klassen unterbunden werden. Diese Entkopplung erleichtert Veränderungen im Rahmen des Entwicklungsprozesses. Die Klasse ist als Singleton implementiert. Das stellt sicher, dass alle Nutzer auf das gleiche Objekt und damit auf die gleichen Werte zugreifen. In der Klasse modelData sind beispielsweise Werte wie die aktuelle Auslenkung der Joysticks, Geschwindigkeit, bereits zurückgelegte Strecke oder ggf. der aktuelle visuelle Hinweis gespeichert. Für jede Information existiert eine eigene Instanzvariable. Das Auslesen oder Setzen eines Wertes erfolgt ausschließlich über Methoden. Die Variablen werden nicht direkt manipuliert. Die Variablen werden in der

Ein Beispiel für die verwendete Singletonimplementierung befindet sich im Anhang A.3.1.

initialize Methode mit einem Standardwert initialisiert. Zwischen den einzelnen Schritten des Experimentes müssen einige Werte, beispielsweise Geschwindigkeit, zurückgesetzt werden. Diese Aufgabe übernimmt die Methode „resetForNewStep“.

4.3.1.5 *Die Klasse OPGuiMasterControl - Der Konfigurationsdialog*

Dieses Element des GUI nimmt eine zentrale Rolle für die Benutzung des Operateursarbeitsplatzes ein. Es wird ausschließlich vom Versuchsleiter eingesetzt, um einen Operateursarbeitsplatz zusammenzustellen. Der Operateur verwendet oder sieht den Konfigurationsdialog nicht.

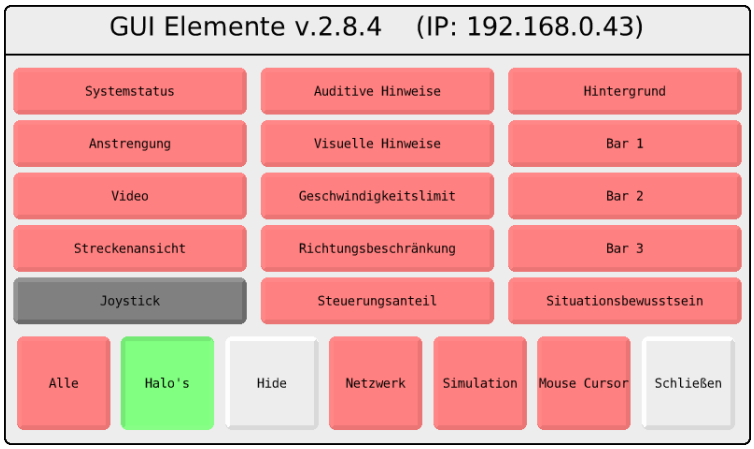

Abbildung 23: Konfigurationsdialog (Version 2.8.4)

Im oberen Bereich des Dialogs befindet sich die 3 x 5 Buttonmatrix. Jedem Button ist einem Teil des Operateursarbeitsplatzes zugeordnet. Dazu zählen auch die schwarzen Balken zur Abtrennung der einzelnen Sektionen des Operateursarbeitsplatzes oder der Hintergrund. Nach dem Starten des Dialogs sind alle Buttons rot gefärbt. Dadurch wird angezeigt, dass die Elemente inaktiv sind. Durch einen Klick auf einen Button wird die dazugehörige Klasse instanziiert, an der voreingestellten Position angezeigt und der Button grün gefärbt. Das kann in beliebiger Reihenfolge und Häufigkeit durchgeführt werden. Es wird auch sichergestellt, dass die Elemente die korrekte Reihenfolge in der Tiefe haben, d.h., sie werden explizit vor dem Hintergrund platziert. Der Konfigurationsdialog liegt immer vor allen anderen Elementen. Von dem eben beschriebenen Verhalten weicht die Anzeige der Joystickauslenkungen in Teilen ab. Da eine einzelne Darstellung nicht vorgesehen ist, ist eine Anzeige der Streckenansicht Vorraussetzung, um den Button überhaupt zu aktivieren. Davon abgesehen verhält sich der Button für die Joystickauslenkungen wie die anderen. Unter der 3 x 5 Matrix liegen sieben weitere quadratische Buttons mit Sonderfunktionen:

ALLE Mit Hilfe dieses Buttons können sämtliche Elemente – bis auf die Joystickanzeige – des Operateursarbeitsplatzes mit einem Klick erstellt oder wieder verworfen werden.

HALOS Während eines Versuchs verwendet der Operateur zur Bedienung des Operateursarbeitsplatzes eine Maus. Ein Standardverhalten für Morphe ist bei Betätigung der dritten Maustaste – im Normalfall das Mausrad – ein sogenanntes Halo-Menü anzuzeigen. Durch dieses Menü kann der betreffende Morph u.a. in seiner Größe verändert, gedreht, verschoben und auch gelöscht werden. Das alles sind Interaktionsmöglichkeiten, die dem Operateur aufgrund der Versuchsanordnung nicht zur Verfügung stehen sollten. Daher wurde in der Klasse OPWindowRectangleMorph der entsprechende Eventhandler überschrieben. Dadurch zeigt ein Klick auf die dritte Maustaste das Halo-Menü nur noch bei aktiviertem Halo-Button. Während der Entwicklung des Operateursarbeitsplatzes sind die Funktionen des Menüs hilfreich. Aus diesem Grunde wird das Halo-Menü nicht komplett deaktiviert. Vor einem Versuch schaltet der Versuchsleiter mit Hilfe dieses Buttons das Halo-Menü für den Operateursarbeitsplatz vollständig aus.

HIDE Während des Versuchs soll der Konfigurationsdialog nicht angezeigt werden. Für den Versuch ist er trotzdem notwendig, da er unverzichtbare Funktionaltitäten wie das Stepping bereitstellt. Aus diesem Grund wird er für die Versuchsdauer mit Hilfe dieses Buttons ausgeblendet. Durch Betätigen der Leertaste kann er wieder eingeblendet werden. Da die Tastatur dem Operateur nicht zur Verfügung steht, ist auch ausgeschlossen, dass der Dialog vorzeitig eingeblendet wird.

MOUSE CURSOR Für den Versuch wird ein anderer Mauszeiger verwendet. Durch diesen Button kann der Versuchsleiter zwischen dem Standardmauszeiger und der individuellen Variante wechseln.

NETZWERK UND SIMULATION Diese beiden Buttons können nicht zur gleichen Zeit aktiviert werden, da ihre Funktionen im Konflikt zueinander stehen. Beim Start des Konfigurationsdialogs sind beide Buttons rot gefärbt. Wird einer der beiden Buttons geklickt, wird der andere ausgegraut und ist nicht länger wählbar. Erst wenn der erste Button erneut geklickt und die damit verbundene Funktionalität beendet wird, werden beide Buttons wieder rot gefärbt. Beide Buttons aktivieren Funktionen, die den Operateursarbeitsplatz mit Eingaben versorgen, z.B. welche Strecke gefahren wird oder die Geschwindigkeit des Trackingobjektes. Der Button „Netzwerk“ aktiviert die Übertragung der

MWB Eingaben, die von der Trackingskomponente SAM über das Netzwerk an den Operateursarbeitsplatz übermittelt werden. Der Button „Simulation“ erzeugt künstlich die gleichen Informationen und bestückt mit ihnen dieselben Bereiche im Datenspeicher modelData. Aus diesem Grunde dürfen die beiden Buttons nicht gleichzeitig aktiviert werden.

VERSIONINFORMATION IN DER TITELZEILE Squeak ist zwar mit einer Historisierungsfunktion versehen, dennoch sind einzelne Releases nicht ohne weiteres unterscheidbar. Daher wurde ein Klasse OPVersion eingeführt. Diese hat nur die Aufgabe, die Versionsnummer vorzuhalten. Der Entwickler muss die Versionsnummer kontinuierlich pflegen. Dadurch wird der Umgang und die Unterscheidung des Arbeitsstandes jedoch deutlich erleichtert.

IP ADRESSE IN DER TITELZEILE Für den Versuchsaufbau ist es notwendig, dass die Rechner im Netzwerk miteinander verbunden werden. Dazu werden den einzelnen Komponenten des ATEO-Testsystems gegenseitig die IP-Adressen bekanntgemacht. Um diesen Vorgang zu erleichtern und zu beschleunigen, wird in der Titelzeile die aktuelle IP Adresse angezeigt.

SCHLIESSEN Dieser Button schließt den Konfigurationsdialog. Sämtliche noch vorhandenen Elemente des Operateursarbeitsplatzes werden gelöscht. Um Fehlbedienungen zu reduzieren, wird der Nutzer bei Betätigung des Buttons durch einen weiteren Dialog zur Bestätigung aufgefordert.

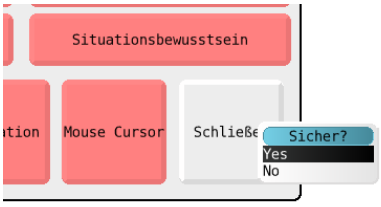

Abbildung 24: Sicherheitsabfrage beim Schliessen des Konfigurationsdialogs (Version 2.8.4)

Die Hauptprogrammschleife des Operateursarbeitsplatzes ist Teil des Konfigurationsdialoges. Für die zyklische Verarbeitung wird Stepping benutzt. Beim Stepping handelt es sich um eine Funktionalität von Morphen. Damit lässt sich ein Morph takten. In jedem Takt wird eine besondere Methode aufgerufen und die dort definierten Instruktionen abgearbeitet. Dazu müssen folgende Schritte durchgeführt werden:

1. Implementieren der Methode „stepTime“. Die Methode gibt die Anzahl der Millisekunden zwischen zwei Steps zurück.
2. Implementierung der Methode „step“. Diese Methode wird bei aktiviertem Stepping periodisch aufgerufen und die dort hinterlegte Funktionalität abgearbeitet.
3. Aufruf der Methode „startStepping“ am Morph: „self startStepping“

Diese Funktionalität wird im Konfigurationsdialog verwendet, um einen potentiell unendlichen Zyklus zu starten. In jedem einzelnen Zyklus findet eine Verarbeitung des Operateursarbeitsplatzes statt. Das umfasst:

- Überprüfung auf neue Netzwerkdaten, deren Verarbeitung und Bereitstellung in der modelData erfolgt
- Aktualisierung der angezeigten Einzelelemente, z.B. Strecke, Anstrengung oder Joystickauslenkung
- Versand von Daten an SAM, z.B. das Geschwindigkeitslimit, gewählte auditive Hinweise oder mögliche Richtungsbeschränkungen

4.3.2 *Operateursarbeitsplatz in Ausbaustufe 1*

Der Operateur hat in dieser Ausbaustufe folgende Elemente zu seiner Verfügung: Das Videobild der MWB, die Streckenansicht, den Systemstatus sowie die visuellen und die auditven Hinweise.

4.3.2.1 *Videobild der Personen*

INFORMATION	INHALT
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	0
Anzahl Klassenvariablen	0
verwendete Klassen	VPVideoMorph

Tabelle 1: Informationen zum Element Videobild

AUFGABE In diesem Element des Operateursarbeitsplatzes ist ein Videobild der beiden MWB zu sehen. Das Videobild wird in Echtzeit übertragen. Eine Zuordnung der MWB ist mit Hilfe entsprechender Symbole über dem Videobild möglich.

Abbildung 25: Videobild

DETAILS DER IMPLEMENTIERUNG Diese Komponente dient der Wiedergabe eines Videobildes der MWB. Squeak selbst verfügt nicht über eine derartige Funktionalität. Die Bereitstellung erfolgt mit Hilfe eines Plugins. In der vorhergehenden Studienarbeit [12] fand eine Analyse für ein Versionsupdate statt. Dabei wurde festgestellt, dass u.a. aufgrund der Inkompatibilität des Kamera Plugins darauf verzichtet werden sollte.

4.3.2.2 *Streckenansicht*

INFORMATION	INHALT
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	12
Anzahl Klassenvariablen	0
verwendete Klassen	OPJoystick

Tabelle 2: Informationen zum Element Streckenansicht

AUFGABE Dieses Fenster stellt dem Operateur einen Ausschnitt der aktuell gefahrenen Strecke zur Verfügung. Der schwarze Rahmen markiert den aktuell für die MWB sichtbaren Streckenabschnitt. Der Bereich darunter dient der Nachschau. Die Vorschau, der größte Bereich der Streckenansicht, befindet sich über dem Rahmen. Durch die Streckenansicht kann der Operateur die ak-

tuelle Situation der MWB und den künftigen Streckenverlauf einsehen. Diese Ansicht ist eine wichtige Informationsquelle für den Operateur zur Führung und Überwachung.

Eine weitere Informationsquelle in diesem Element ist die Darstellung der Joystickauslenkung. Für jeden MWB existiert ein eigenes Element. Diese sind auf der rechten und linken Seite der Streckenanzeige angeordnet. Die beiden Elemente zeigen durch ihre Gestaltung auch die Geschwindigkeit, den aktuell eingestellten Steuerungsanteil und bei Aktivierung die Richtungsbeschränkung an.

DETAILS DER IMPLEMENTIERUNG

Strecke In der Komponente SAM wird die Strecke als ein großes Bitmap geladen und mittels blitting bewegt. Dem Operateur wird während einer Fahrt zeitgleich die gleiche Strecke in kleinerer Grösse angezeigt. Eine geringe Verzögerung resultiert aus der Übertragung der benötigten Daten über das Netzwerk. Die im Operateursarbeitplatz verwendeten Streckengrafiken sind die skalierte Versionen der in SAM eingesetzten Streckenbilder. Vor einer Fahrt wird der Name der anzuzeigenden Strecke über das Netzwerk an den Operateursarbeitsplatz übertragen. Danach wird das skalierte Streckenbild als Morph geladen.

Dieser Streckenmorph wird dem GUI Element „Streckenansicht“ als Submorph hinzugefügt. Um die Strecke zu bewegen, wird die Position des Streckenmorphs in der y-Achse verändert. Durch das Netzwerk wird in jedem Step die durch die MWB zurückgelegte Schrittweite übermittelt. Ein Step beschreibt eine Zeitspanne von 39 ms. Die maximale Schrittweite in der Trackingkomponente SAM liegt bei 20.48 Pixel. Die Darstellung der Strecke verwendet den Skalierungsfaktor von 0.375. Die maximale Schrittweite auf dem Operateursarbeitsplatz beträgt daher 7.68 Pixel.

Die Problematik der Nachkommastellen bei Pixeln wird im Abschnitt 5.6 auf Seite 73 besprochen.

Beim Streckenwechsel werden einige Variablen zurückgesetzt und eine neue Streckengrafik geladen.

Schweif Der Schweif liefert eine visuelle Historie der Geschwindigkeit und der horizontalen Auslenkungen des Trackingobjektes. Der Schweif besteht aus einer Anzahl von Morphen in gleicher Grösse wie der Trackingmorph. In jedem Step wird die horizontale Auslenkung und die Schrittweite ausgewertet. Diese beiden Werte bestimmen die Position und die Färbung des aktuellen Schweifmorphes. Die Leistung der Streckenansicht kann unter einer zu großen Anzahl von Morphen einbrechen. In jedem Step wird ein neuer Morph erzeugt. Daher ist es für eine stabile Performance der Streckenansicht notwendig, die nicht länger im sichtbaren Bereich befindlichen Morphe zu löschen. Durch die

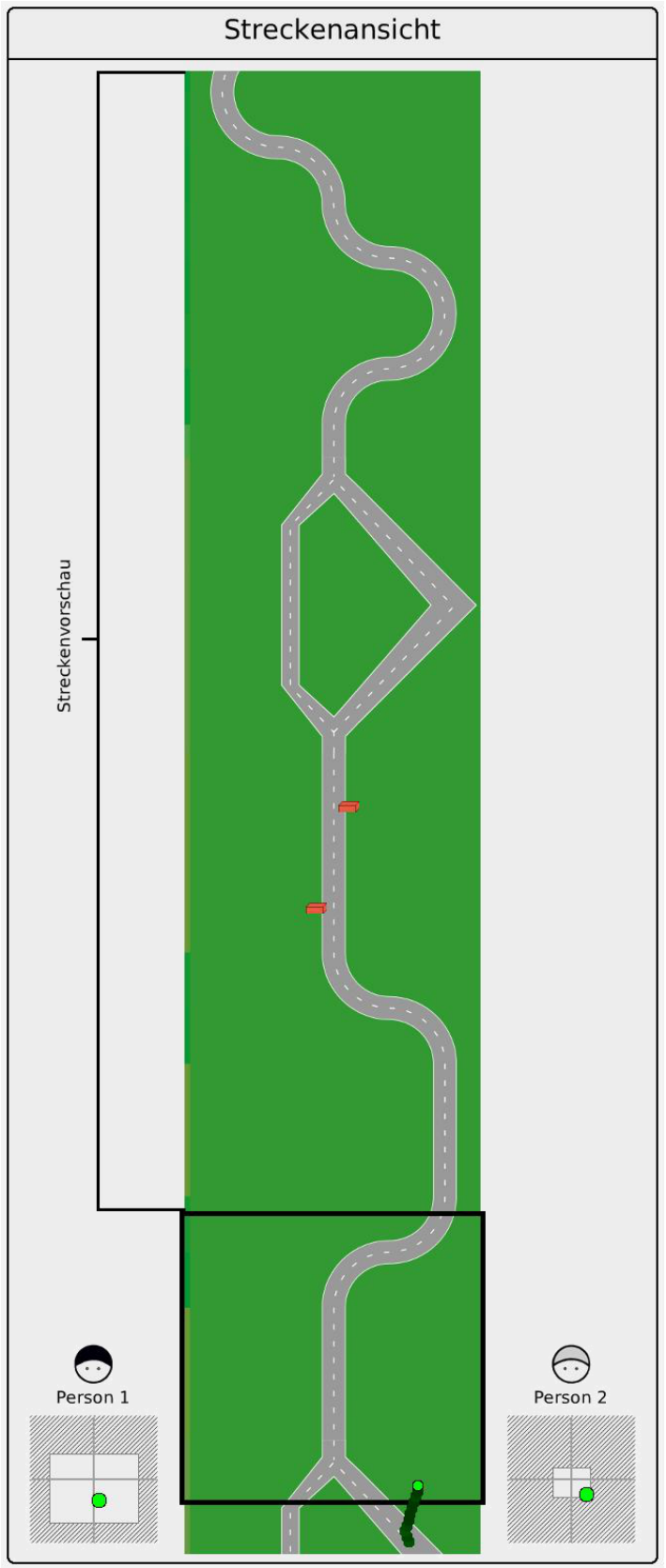

Abbildung 26: Streckenansicht mit Elementen zur Darstellung der Joystickauslenkungen

Implementierung wird sichergestellt, dass zu jeder Zeit maximal die 10 aktuellsten Schweifmorphe existieren.

Die Schweifmorphe sind in einem eigenen Morph eingebettet. Dieser Morph liegt über der angezeigten Strecke, ist transparent und exakt gleich groß. Seine einzige Aufgabe ist die Aufnahme der Morphe, die den Schweif bilden. Dieser Containermorph wird in jedem Step um die gleiche Schrittweite wie der Streckenmorph bewegt. Dadurch wird in jedem Step ein neuer Schweifmorph erstellt und auf dem transparenten Containermorph als Submorph hinzugefügt. Weil der Container der Schrittweite entsprechend weiterbewegt wird, entsteht der Eindruck das Trackingobjekt würde einen Schweif hinter sich herziehen. Jeder dieser Morphe bildet eine Momentaufnahme der horizontalen Auslenkung und der Schrittweite. Dadurch wird auf der Streckenanzeige ein „Schweif“ des Trackingobjektes erstellt.

Hindernisse In Abhängigkeit der Streckenkonfiguration können sich auf einer Strecke Hindernisse befinden. Diese erscheinen als rote Objekte auf der Strecke. Die Anzahl, Art und Position der Hindernisse für eine Strecke sind in einer Textdatei gespeichert. Für jede Strecke existiert eine eigene Datei, da die Platzierung der Hindernisse in Abhängigkeit der Strecke vorgenommen wird. Die Datei enthält für jedes Hindernis einen Eintrag. Dieser besteht aus zwei Zahlen und nimmt eine Zeile in der Datei ein. Die erste Zahl stellt die Art des Hindernisses dar. Durch ein Semikolon getrennt folgt ein die zweite Zahl. Diese ist der Offset und beschreibt den Abstand eines Hindernisses vom unteren Rand der Strecke. Die Textdatei wird beim Erstellen der Strecke in SAM verarbeitet. Entsprechend der Informationen werden die Hindernisse auf der Strecke platziert. Über das Netzwerk wird der Inhalt der Textdatei an den Operateursarbeitsplatz übertragen.

Mehr Details zur Logdatei oder der Konfigurationsdatei der Hindernisse können der Spezifikation [4] entnommen werden.

Im ersten Schritt werden die Hindernisse auf Grundlage der empfangenen Information als Morphe erstellt. In einer Collection „obstacles“ (Instanzvariable) werden die Hindernisse gespeichert. Das ermöglicht auf einfache Weise in jedem Step über alle Hindernisse zu iterieren und nicht länger sichtbare Hindernisse zu löschen. Zusätzlich wird diese einfache Art des Zugriffs in jedem Step verwendet, um die notwendigen Informationen für die Trackeditvariable des Logs zusammenzustellen. Dadurch ist es auch möglich, die Bewegungen der dynamischen Hindernisse zu verarbeiten und anzuzeigen.

Zusätzlich werden die Hindernisse als Submorphe der Strecke hinzugefügt. Dadurch werden sie beim Bewegen der Strecke ebenfalls bewegt. Die Positionen der Hindernisse müssen also nicht explizit angepasst werden.

Joystickauslenkungen Die beiden Anzeigen der Joystickauslenkungen sind optionale Elemente der Streckenansicht. Eine Einzelanzeige für nur einen MWB ist nicht vorgesehen. In der aktuellen Version 2.8.4 werden immer beide Joystickauslenkungen angezeigt.

Das Anzeigeelement für die Joystickauslenkungen ist als eigene Klasse realisiert. Da die beiden Elemente fast identisch sind, lag es nahe zum Zwecke der Codereduzierung eine eigene Klasse zu erstellen. Das MWB Symbol und seine Beschriftung sind nicht Teil davon und werden als Parameter übergeben. In dieser Klasse sind Methoden zur Erstellung und Verarbeitung der neuen Positionen zusammengefasst. Die Klasse OPJoystick erbt direkt von der Klasse ImageMorph. Diese Klasse erbt direkt von der Klasse Morph. Sie wird für Morphrepräsentationen von Bilddateien verwendet.

Ein Joystickanzeige besteht aus fünf übereinander angeordneten Morphen. Die schraffierte Fläche bildet den Hintergrund und ist die Bilddatei der Klasse OPJoystick selbst. Dazu kommen noch weitere vier Morphe, die als Submorphe dem Imagemorph zugeordnet werden. Die beiden Hilfslinien, die die viereckige Grundfläche in Quadranten teilt, sind RectangleMorphe. Das grüne Objekt, welches die aktuelle Auslenkung des entsprechenden MWB darstellt, ist wie das Trackingobjekt ein EllipseMorph. Der weiße RectangleMorph kennzeichnet den Bereich, in dem die Auslenkungen in die Berechnung für die neue Position des Trackingobjektes einfließen. Die Anzeige zur momentanen Auslenkung wird nur auf dem eben beschriebenen weißen RectangleMorph platziert.

Im Operateursarbeitsplatz existieren drei Elemente, welche die Darstellung der Joystickauslenkungen beeinflussen. Das sind die Elemente Geschwindigkeitslimit, Richtungsbeschränkung und Steuerungsanteil. Einschränkungen durch eines der vorgenannten Elemente, werden anhand der weißen Fläche an den Operateur zurück gemeldet. Die Abbildungen 27-30 auf Seite 55 zeigen einige Darstellungsmöglichkeiten der beiden Elemente. Für weitere Erläuterungen siehe die Studienarbeit [12].

4.3.2.3 *Systemstatus*

AUFGABE In diesem Element kann der Operateur verschiedene Informationen ablesen. Aktuell sind das die jeweiligen Strategien der MWB oder die Nummer der aktuellen Fahrt. Perspektivisch könnte es noch um weitere Informationen ergänzt werden. Denkbar wären beispielsweise ein Fehlermaß oder die Zeit der bisher abgeschlossenen Fahrten.

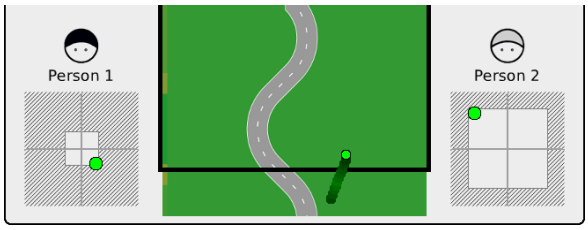

Abbildung 27: Steuerungsanteil 30/70, Geschwindigkeit 100%, keine Richtungseinschränkung

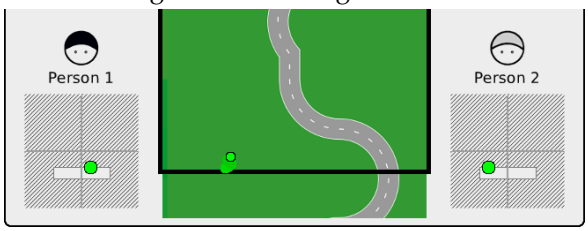

Abbildung 28: Steuerungsanteil 50/50, Geschwindigkeit 20%, keine Richtungseinschränkung

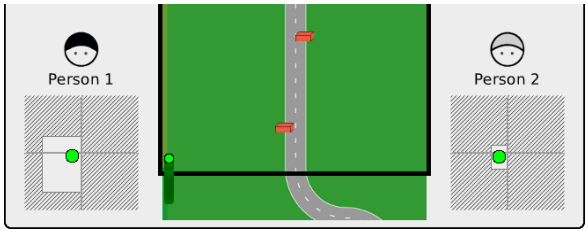

Abbildung 29: Steuerungsanteil 70/30, Geschwindigkeit 70%, Richtungseinschränkung auf rechts

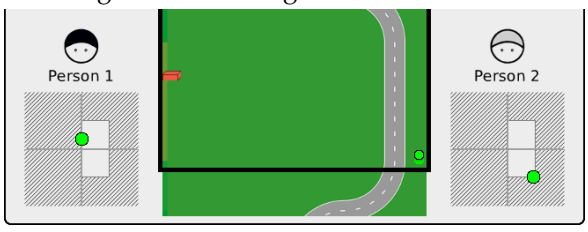

Abbildung 30: Steuerungsanteil 50/50, Geschwindigkeit 100%, Richtungseinschränkung auf links

INFORMATION	INHALT
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	1
Anzahl Klassenvariablen	0
verwendete Klassen	-

Tabelle 3: Informationen zum Element Systemstatus

Abbildung 31: Element Systemstatus

DETAILS DER IMPLEMENTIERUNG Das Fenster Systemstatus ist direkt von der Klasse OPWindowRectangleMorph abgeleitet. Die Erstellung der MWB-Symbole mit den Beschriftungen darüber und darunter wurde in eine Methode ausgelagert. Dadurch wurde doppelter Code vermieden. Die Individualisierung

wird durch die Übergabe von Parametern beim Methodenaufruf erreicht.

Im unteren Teil des Elementes wird die Nummer der aktuellen Fahrt angezeigt. Die erste Zahl ist ein Stringmorph, der als Instanzvariable definiert ist. Mit Hilfe der Methode „setStepNr:“ wird der Inhalt dieses Stringmorphes geändert. SAM übermittelt vor jeder Fahrt über das Netzwerk die Nummer für die nächste Fahrt. Dieser Wert wird als Parameter beim Aufruf der beschriebenen Methode „setStepNr:“ verwendet, um die Anzeige im Fenster Systemstatus anzupassen.

4.3.2.4 *Visuelle Hinweise*

INFORMATION	INHALT
Klassenname	OPVisualHints
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	2
Anzahl Klassenvariablen	0
verwendete Klassen	-

Tabelle 4: Informationen zum Element Visuelle Hinweise

AUFGABE Mit Hilfe des Elementes visuelle Hinweise kann der Operateur in das Tracking eingreifen. Diese Eingriffe sind für die MWB nicht bindend. Sie haben lediglich Vorschlagscharakter. Aus diesem Grund werden sie auch weiche Eingriffe genannt.

Die Hinweise werden sowohl den MWB auf der Strecke, als auch dem Operateur in der Streckenansicht angezeigt. Die visuellen Hinweise können nicht gezielt an einen MWB adressiert werden.

Dem Operateur stehen dazu zwei verschiedene Arten von Hinweisen zur Verfügung: Warnungen sowie Fahrtrichtung & Geschwindigkeit. Durch die Streckenansicht hat der Operateur die Möglichkeit, Hindernisse und Gabelungen vor den MWB zu erkennen. Somit kann er den beiden MWB vorrausschauend visuelle Hinweise geben, bevor diese Hindernisse oder Gabelungen für sie sichtbar sind. Zusätzlich zu den Warnungen kann der Operateur weitere Hinweise geben. Diese schlagen den MWB konkret eine Richtung der Joystickauslenknung vor, d.h. Links- oder Rechtsauslenkung oder eine Anpassung der Geschwindigkeit durch Auslenkungen auf der vertikalen Achse.

Das Element ist auf der Abbildung 16 auf Seite 37 zu finden.

DETAILS DER IMPLEMENTIERUNG Die Implementierung unterteilt sich im Wesentlichen in vier Bereiche. Das Erstellen der

Buttons, Verarbeitung der Klicks, Erstellen und Anzeige der Rückmeldungen.

Jeder Hinweis, den der Operateur geben kann, erhält einen eigenen Button. Jede Kategorie von visuellen Hinweisen ist in einem eigenen Element gruppiert. Dies wird mit Hilfe der Einbettung in einem hellen Morph kenntlich gemacht. Die Erstellung der entsprechenden Elemente wurden jeweils in eigene Methoden ausgelagert, um die Übersichtlichkeit der initialize Methode zu erhöhen. Obwohl die Implementierung der beiden Methoden sehr ähnlich ist, erfüllen sie jeweils eine andere Aufgabe. Bei beiden wird ein Morph erstellt, der die anderen Bestandteile der Hinweiskategorien aufnimmt. Er übernimmt die Funktion eines Containers. Dem Containermorph werden dann die benötigte Anzahl von Buttons und der Titel des Elementes hinzugefügt. Die Buttons werden unter der Verwendung der Methode „createIconicButton: withActionSelector: andSubTitle:“ erstellt. Die Methode ist in der Oberklasse OPWindowRectangleMorph implementiert. Um die häufig benötigte Funktionalität zur Erstellung eines Rectanglemorphes mit einer Layout Policy nicht in verschiedenen Klassen implementieren zu müssen, wurde auch dafür eine eigene Methode „createRectangleMorphWithLayoutPolicy: extent:“ implementiert.

Für die Verarbeitung der Buttonklicks sind eigene Methoden verantwortlich. Jeder Button verfügt über einen Actionselector. Diese sind relativ einfach aufgebaut und erfüllen zwei Aufgaben. Sie schreiben je nach betätigtem Button einen entsprechenden Wert in den Datenspeicher modelExport, wodurch letztlich in SAM die Hinweise angezeigt werden können. Weiterhin rufen sie die Methode „showTrackPreviewHints:“ auf, um die Rückmeldung auf dem Operateursarbeitsplatz in der Streckenansicht anzuzeigen.

Das Vorbereiten und Anzeigen der Rückmeldungen auf dem Operateursabreitsplatz sind jeweils in eigenen Methoden umgesetzt.

Um Verzögerungen während eines Versuches zu vermeiden, werden die Grafiken für die anzuzeigenden Rückmeldungen bereits beim Erstellen des Elementes „Visuelle Hinweise“ geladen. Der Methode „loadTrackPreviewHints:“ wird ein Dictionary als Parameter übergeben. Diese enthält die verwendeten Grafiken. Die Grafiken werden vorher im Zuge der „initialize“ Methode in das Dictionary geladen. Das Dictionary wird ebenfalls bei der Erstellung der Button Elemente für die Warnungen und Hinweise genutzt. Dabei wird auf die Grafiken nur indirekt referenziert, in dem der Zugriff über einen Schlüssel des Dictionarys erfolgt. Damit wird nur an einer Stelle, dort wo das Dictionary mit den Grafiken bestückt wird, überhaupt der Pfad und Dateiname

verwendet. Eventuelle Änderungen, wie das Benutzen anderer Grafiken, sind damit sehr einfach umzusetzen.

Die beiden Instanzvariablen der Klasse „OPHintsVisual“ sind Dictionarys. Sie enthalten die Grafiken die im Element „Streckenansicht“ als Rückmeldung angezeigt werden, wenn die entsprechenden Buttons geklickt werden. In der Methode „loadTrackPreviewHints:“ werden die beiden Instanzvariablen „leftGfx“ und „rightGfx“ mit den Grafiken aus dem als Parameter übergebenen Dictionary bestückt. Die einzelnen Einträge in den beiden Instanzvariablen unterscheiden sich nur in der Position der Grafiken. An diesen Positionen, links und rechts von der Strecke, werden sie bei einem Klick auf den korrespondierenden Button angezeigt. Zusätzlich wird noch ein weiterer Eintrag eingefügt. Dieser enthält keine Grafik sondern ist nur ein einfarbiger Morph. Er dient dazu, bei kurz aufeinander folgenden Klicks ein kurzes Flickern zu erzeugen. Auch die auditiven Hinweise verfügen bei einem Klick auf die Buttons über Rückmeldungen. Die Hinweise für die Joystickauslenkungen sind bei beiden Elementen sehr ähnlich gestaltet. Sie verwenden beispielsweise die gleichen Grafiken. Damit kann es vorkommen, dass nach einem auditiven Hinweis der Operateur zeitnah einen visuellen Hinweis gibt. Auch der umgekehrte Fall ist möglich. Verwenden beide Buttons die gleiche Grafik, da sie die gleiche Bedeutung haben, wie beispielsweise „schneller fahren“, würden die Rückmeldungen einfach übereinander angezeigt. Damit wäre nicht eindeutig erkennbar, ob der letzte Hinweis auch erfolgreich gegeben worden ist. Damit wird das Paradigma durchbrochen, dass jeder steuernde Eingriff im Operateursabreitsplatz auch im Operateursabreitsplatz zurückgemeldet wird. Mit Hilfe des einfarbigen Morphes wird die Anzeige kurz unterbrochen, um das oben erläuterte Flickern zu erzeugen. Das wird durch die Anzeige des „leeren“ Eintrages der Instanzvariablen erreicht, welcher für 150 ms angezeigt wird. Danach wird die Rückmeldung des zuletzt geklickten Buttons gezeigt.

4.3.2.5 *Auditive Hinweise*

INFORMATION	INHALT
Name der Klasse	OPAuditiveHints
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	13
Anzahl Klassenvariablen	0
verwendete Klassen	-

Tabelle 5: Informationen zum Element Auditive Hinweise

AUFGABE Dieses Element ermöglicht es dem Operateur den beiden MWB auditive Hinweise zu geben. Aufbau und Handhabung ähnelt dabei in weiten Teilen den visuellen Hinweisen. Beide MWB tragen während des Versuches Kopfhörer. Dadurch kann der Operateur gezielt bestimmen, wer Adressat eines Hinweises ist. Bei den Hinweisen handelt es sich um im Vorfeld aufgenommene Audiodateien mit entsprechendem Inhalt.

Ein Hinweis hat für die MWB keine sichtbare Komponente. Für den Operateur hingegen gibt es – wie bei den visuellen Hinweisen – Rückmeldungen für ausgelöste Hinweise. Diese werden am gleichen Ort wie die der visuellen Hinweise links und rechts neben der Streckenansicht platziert. Es werden auch die gleichen Grafiken verwendet. Weil bei den auditiven Hinweisen nicht immer beide MWB Empfänger sind, wird dies auch in den Rückmeldungen berücksichtigt. Wird nur ein MWB adressiert, wird auch nur auf der entsprechenden Seite der Streckenansicht die korrespondierende Rückmeldung angezeigt. Wurden beide MWB ausgewählt, so werden Rückmeldungen auf beiden Seiten der Streckenansicht angezeigt, analog zu den visuellen Hinweisen.

Die Auswahl der Adressaten erfolgt über drei Buttons. Diese sind in einer Leiste zusammengefasst, die erst bei einem Buttonklick angezeigt wird. Dabei wird die Leiste immer über dem geklickten Button zentral positioniert. Das Auslösen eines auditiven Hinweises erfolgt in zwei Schritten. Zuerst wird der Hinweis ausgewählt und danach in der dann eingeblendeten Adressauswahlleiste der gewünschte Adressat.

Das Element ist auf der Abbildung 18 auf Seite 38 zu finden.

DETAILS DER IMPLEMENTIERUNG Die Implementierung der auditiven Hinweise ähnelt in Teilen der der visuellen Hinweise. Die grundsätzliche Vorgehensweise bei der Erstellung der einzelnen Elemente ist fast identisch. Ein Unterschied ist die Verwendung von Instanzvariablen für die Buttons. Dieses Vorgehen ist notwendig, um das Auswahlelement für die Adressaten zu positionieren.

Die Implementierung der Rückmeldungen ähnelt der der visuellen Hinweise. Daher soll hier nur auf die Unterschiede eingegangen werden. Der Inhalt des Dictionarys, das in der „initialize“ Methode erstellt wird und sämtliche verwendete Grafiken enthält, unterscheidet sich aufgrund der verschiedenen Grafiken. Für jedes einzelne Element existiert eine eigene Methode zur Erstellung. Die Methode zur Anzeige der Rückmeldungen wertet den Input der Auswahlleiste aus. Die Auswahl wird als Parameter beim Aufruf der Methode „showTrackPreviewHints: aHintString forMwis:“ übergeben. In Abhängigkeit des ausgewählten Adressaten werden nur die dazugehörigen Rückmeldungen über den MWB Symbolen der Streckenansicht gezeigt.

Um einen auditiven Hinweis in SAM abzuspielen, werden zwei Informationen benötigt. Zum einen die Information, um welchen Hinweis es sich handelt und zum anderen wer ihn hören soll.

In den Methoden, die die Klicks der Hinweisbuttons verarbeiten, wird die Information um welchen auditiven Hinweis es sich handelt in modelExport geschrieben. Zusätzlich wird die Auswahlleiste für die Adressaten eingeblendet. Diese wird immer zentriert über den geklickten Button angezeigt. Damit liegt der mittlere Button der Auswahlleiste für die Adressaten deckungsgleich über dem geklickten Hinweisbutton. Bei einem Klick auf einen Button der Adressatenauswahlleiste wird die Information, an wen der Hinweis adressiert ist, in der modelExport hinterlegt. Die Auswahlsequenz ist damit abgeschlossen. Die Auswahlleiste wird wieder ausgblendet.

Zwischen zwei zu fahrenden Strecken müssen eventuelle Veränderungen am Operateursarbeitsplatz auf den Initialzustand zurückgesetzt werden. Sollte der Operateur am Ende einer Fahrt einen Hinweisbutton geklickt haben, ohne anschließend einen Adressaten auszuwählen, muss die Auswahlleiste der Adressaten ausgeblendet werden. Das ist erforderlich, um den Initialzustand des Elementes wieder herzustellen. Zu diesem Zweck wurde eine eigene Methode implementiert.

4.3.3 *Operateursarbeitsplatz in Ausbaustufe 2*

Zusätzlich zu den Elementen der ersten Ausbaustufe stehen dem Operateur in dieser Ausbaustufe die zusätzlichen Elemente Steuerungsanteil, Richtungsbeschränkung und Messung des Situationsbewusstseins zur Verfügung.

4.3.3.1 *Steuerungsanteil*

INFORMATION	INHALT
Name der Klasse	OPInputDistribution
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	10
Anzahl Klassenvariablen	0
verwendete Klassen	-

Tabelle 6: Informationen zum Element Steuerungsanteil

AUFGABE Mit diesem Element kann der Operateur den Steuerungsanteil zwischen den MWB verschieben. Beide Anteile ergeben zusammen zu jeder Zeit 100%. Wenn beispielsweise ein MWB

seiner Instruktion nicht folgt oder die Anstrengung des MWB zu hoch ist, könnte der Operateur den Steuerungsanteil dieses MWB reduzieren. Um in die Ausgangsverteilung zurückzukehren, wurde ein Button erstellt, der beiden MWB einen Anteil von 50% zuweist.

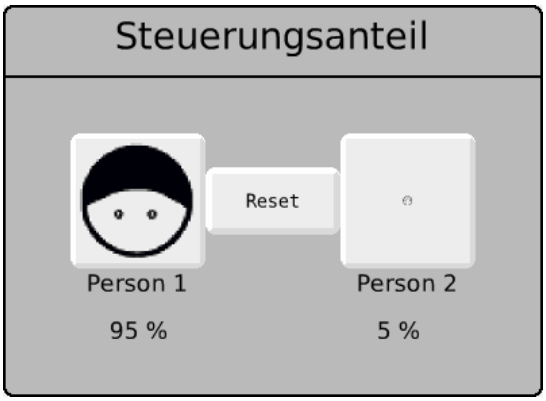

Abbildung 32: Element Steuerungsanteil, Verteilung 95/05

DETAILS DER IMPLEMENTIERUNG Details zur Implementierung wurden bereits in der vorhergehenden Studienarbeit [12] erläutert und sollen hier nicht wiederholt werden.

4.3.3.2 *Richtungsbeschränkung*

INFORMATION	INHALT
Name der Klasse	OPDirectionButtons
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	2
Anzahl Klassenvariablen	0
verwendete Klassen	-

Tabelle 7: Informationen zum Element Richtungsbeschränkung

AUFGABE Der Operateur kann mit Hilfe dieses Eingriffes einen Teil der horizontalen Joystickauslenkungen von der Berechnung der Position des Trackingobjektes ausschließen. Wahlweise kann er entweder die Links- oder der Rechtsauslenkung blockieren. Für die MWB bedeutet das, dass die Auslenkungen in die blockierte Richtung keine Wirkung zeigt. Das Trackingobjekt läßt sich nicht mehr in die blockierte Richtung lenken. Es kann immer nur eine Einschränkung aktiviert sein. Erneutes Klicken auf eine aktivierte Beschränkung verändert die Einstellungen nicht. Klickt der Operateur auf einen Button für eine Einschränkung und

die andere ist bereits aktiv, so wird die geklickte gültig und die vorherige zurückgesetzt.

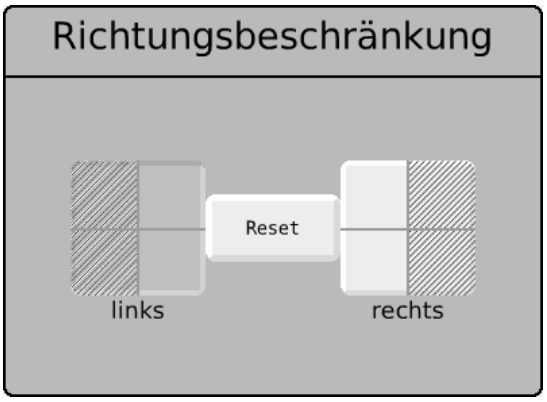

Abbildung 33: Element Richtungsbeschränkung, Linksauslenkungen blockiert

DETAILS DER IMPLEMENTIERUNG Die beiden Buttons werden mit einer Methode der Klasse OPWindowRectangleMorph erstellt. Es handelt sich dabei um Iconic Buttons, welche an Stelle einer Beschriftung eine Grafik anzeigen. Die Grafik und die Beschreibung unter dem Button werden als Parameter übergeben.

Die beiden äußeren Buttons sind Instanzvariablen. Dadurch können sie aus Eventhandlern im Aussehen angepasst werden. Die durch einen Klick ausgelösten äußerlichen Anpassungen der Farbe und Art der Buttonbegrenzung sind kein Standardverhalten von Buttons, sondern wurden vom Autor implementiert.

Weiterhin exportieren die Eventhandlermethoden auch die vom Operateur eingestellten Einschränkungen nach modelExport. Auch für die Rückmeldungen in der Anzeige der Joystickauslenkungen werden die Informationen bereitgestelt.

Auch für dieses Element ist eine Methode nötig, die zwischen zwei Strecken den Ausgangszustand wieder herstellt.

4.3.4 *Operateursarbeitsplatz in Ausbaustufe 3*

In der dritten Ausbaustufe wird der Operateursarbeitsplatz um die Elemente Anstrengung, Geschwindigkeitslimit und Messung Situationsbewusstsein erweitert. Das ist die derzeit maximale Anzahl von Elementen.

4.3.4.1 *Anstrengung*

AUFGABE In diesem Element wird die subjektive Anstrengung der beiden MWB dargestellt. Die MWB haben nach jeder Fahrt die Möglichkeit, ihre subjektiv empfundene Anstrengung mit Hilfe einer Skala zu bewerten. Diese Information wird vor dem

INFORMATION	INHALT
Name der Klasse	OPMentalWorkload
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	3
Anzahl Klassenvariablen	0
verwendete Klassen	-

Tabelle 8: Informationen zum Element Anstrengung

Beginn der nächsten Fahrt an den Operateursarbeitsplatz über das Netzwerk übertragen und dort angezeigt. Für die Anzeige werden die üblichen MWB Icons genutzt. Zur Rückmeldung werden diese rot gefüllt. Je höher die angegebene Anstrengung, desto höher der Füllstand. Zusätzlich steht unter jedem MWB Icon die Anstrengung in Prozent, wobei 100% den Höchstwert und damit die größtmögliche Anstrengung darstellt. Der Operateur soll durch die Informationen dieses Elementes zu einer besseren Einschätzung darüber gelangen, welche MWB noch über Reserven hinsichtlich Belastbarkeit verfügen. Ein MWB mit einer hohen Anstrengung setzt die Hinweise des Operateur wahrscheinlich weniger oder gar nicht mehr um.

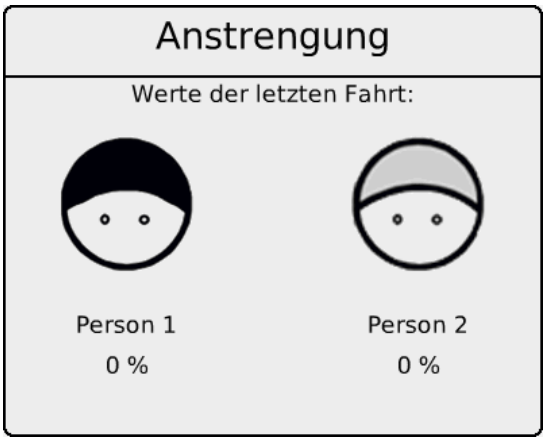

Abbildung 34: Element Anstrengung, initialer Zustand

DETAILS DER IMPLEMENTIERUNG Die Klasse verfügt über drei Instanzvariablen. Die Variable „gaugeColor“ enthält den Farbton für die Färbung der MWB Icons. Die anderen beiden enthalten die Morphe mit den MWB Icons und der Anstrengungsskala. Die Erzeugung dieser Morphe übernimmt eine Methode. Die Individualisierung wird durch Parameter beim Aufruf der Methode gesteuert. Der Rückgabewert wird dann in den Variablen „leftMWI“ und „rightMWI“ gespeichert.

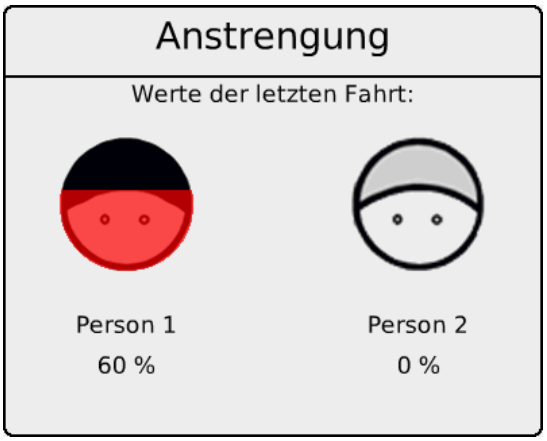

Abbildung 35: Element Anstrengung,
MWB 1 Anstrengung von 60%

Die Darstellung des Füllstandes wird durch das Zusammenspiel zweier Morphe erreicht. Als Container wird zuerst ein viereckiger Morph, ein RectangleMorph, erstellt. Bei der Erstellung wird die Eigenschaft „clipSubmorphs: „ auf true gesetzt. Dadurch wird von dem danach hinzugefügten runden EllipseMorph, ein Submorph des Containers, nur das angezeigt, was sich innerhalb der Grenzen des Containers befindet. Effektiv lassen sich so Submorphen teilweise oder ganz ausblenden. Der EllipseMorph erhält den in der Instanzvariable „gaugeColor“ gespeicherten Farbwert. Die Sichtbarkeit wird über die Höhe des Containers gesteuert. Das Vorgehen ähnelt dem Öffnen einer Jalousie, um nur den gewünschten dahinterliegenden Teil sichtbar zu machen. Eine Anstrengung von 0% entspricht damit einer Höhe des Containers von 0, eine Anstrengung von 30% würde das MWB Icon von unten beginnend zu etwa einem Drittel mit roter Färbung auffüllen.

4.3.4.2 *Geschwindigkeitslimit*

INFORMATION	INHALT
Name der Klasse	OPSpeed
erbt direkt von	OPWindowRectangleMorph
Anzahl Instanzvariablen	1
Anzahl Klassenvariablen	0
verwendete Klassen	OPAdvancedSlider

Tabelle 9: Informationen zum Element Geschwindigkeitslimit

AUFGABE Mit Hilfe des Geschwindigkeitslimits kann der Operateur die potentielle Höchstgeschwindigkeit der MWB einstellen.

Ein Klick auf den Button Reset stellt den Ausgangswert von 100% wieder her.

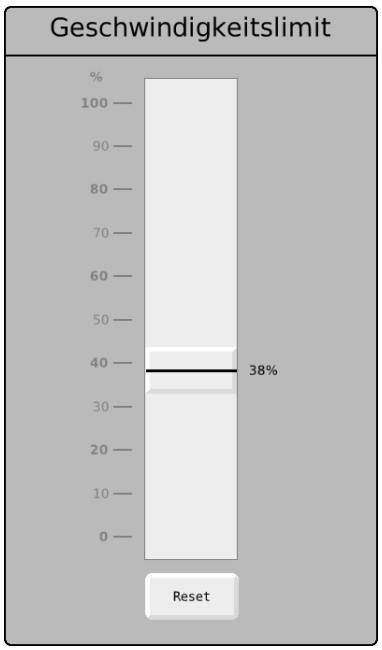

Abbildung 36: Element Geschwindigkeitslimit, Limitierung auf 38% der Höchstgeschwindigkeit

DETAILS DER IMPLEMENTIERUNG In der Klasse OPSpeed wird für die Erstellung des Sliders die Klasse OPAdvancedSlider instanziiert. Diese Klasse ihrerseits erbt von der Standard-Squeak Klasse SimpleSliderMorph. Die Erstellung einer eigenen Schieberegler-Klasse war notwendig, da die Standardimplementierung von Schiebereglern in Squeak einige Einschränkungen aufwies. So lässt sich beispielsweise die Stärke des Sliders nicht explizit bestimmen. Stattdessen wird sie immer in Abhängigkeit der Größe des gesamten Schiebereglers ermittelt. Auch ist der Stringmorph mit der aktuell eingestellten Geschwindigkeit kein Feature der Standardimplementierung. Nur mit Hilfe der Klasse OPAdvancedSlider konnten die Anforderungen an den Schieberegler vollständig umgesetzt werden. Die Skala auf der linken Seite des Schiebereglers wird durch Methoden in der Klasse OPSpeed erstellt. Bei der Umsetzung wurde versucht, auf maximale Flexibilität zu achten. Bei Änderungen in der Skalendarstellung, z.B. 150% anstelle von 100% Maximum oder eine andere Schrittweite der Skala, sind nur wenige Änderungen notwendig.

4.3.4.3 *Messung Situationsbewusstsein*

AUFGABE Eines der Ziele der Versuche ist auch die Bestimmung des Situationsbewusstseins der Operateure. Im Normalfall ist das Element sichtbar, aber ausgegraut und damit nicht für

INFORMATION	INHALT
Name der Klasse	OPSituationAwareness
erbt direkt von	RectangleMorph
Anzahl Instanzvariablen	15
Anzahl Klassenvariablen	0
verwendete Klassen	-

Tabelle 10: Informationen zum Element Messung Situationsbewusstsein

den Operateur benutzbar. Nach dem Erreichen einer vorher festgelegten Strecke wird das Element aktiviert. Während seiner Aktivierung wird eine Reihe von Zuständen durchlaufen:

1. Ein Hinweiston wird abgespielt.
2. Der linke Button ist klickbar und grün gefärbt. Der rechte Button ist hingegen nicht klickbar und grau gefärbt. Ein Timeout startet im Hintergrund, in den meisten Fällen mit einer Dauer von 10 Sekunden. Innerhalb dieser Zeit muss der Operateur den linken Button geklickt haben. Andernfalls kehrt das gesamte Element in seinen Ausgangszustand zurück.
3. Nach dem Klick auf den linken Button wird eine vorher aufgenommene Frage abgespielt, z.B. „Welche Person reagiert momentan schneller auf ihre Hinweise? Eins oder zwei?“. Der linke Button wird deaktiviert und grau gefärbt, der rechte Button aktiviert und grün gefärbt. Im Hintergrund startet erneut ein Timeout. Standarddauer für den zweiten Timeout sind 30 Sekunden. Die Antwort wird vom Versuchsleiter notiert, die Buttonklicks und die benötigte Zeit an SAM übermittelt und dort in das Logfile geschrieben.

Abbildung 37: Element Messung Situationsbewusstsein, Ausgangszustand

DETAILS DER IMPLEMENTIERUNG Das Element Messung Situationsbewusstsein erbt nicht wie die restlichen Elemente von der Klasse OPWindowRectangleMorph, sondern direkt von der Standardklasse RectangleMorph. Damit entfällt die Möglichkeit

auf Methoden wie z.B. für die Erstellung von Buttons, Farbgebung oder Stringmorphe zurückzugreifen. Die konkreten Implementierungen orientierten sich an den Methoden aus OPWindowRectangleMorph, um ein einheitliches Aussehen zu gewährleisten. Die Positionierung erfolgt zentral durch das Hauptsteuerelement OPGuiMasterControl. Die Konfiguration wird in einer Textdatei „situationAwareness.txt“ gespeichert und zeilenweise interpretiert:

Listing 1: Auszug der Konfigurationsdatei „situationAwareness.txt“

```
9; 66000; 10; 30; frage_1.mp3
10; 106000; 10; 30; frage_2.mp3
11; 75000; 10; 30; frage_3.mp3
```

Die erste Zahl entspricht der Schrittnummer gemäß der Versuchskonfiguration in der steps.txt. Die zweite Zahl stellt die zu fahrende Distanz auf der Strecke dar. Nach dem Erreichen dieser Distanz wird das Element aktiviert. Die nächsten beiden Zahlen sind die Timeouts in Sekunden für die Buttonbetätigung. Die erste Zahl entspricht dem linken Button und die zweite dem rechten. Als letztes wird der Dateiname der vorher aufgenommenen Frage angegeben. Die Dateien werden aus dem Ordner „\sa\audio“ geladen.

4.4 TEST DER IMPLEMENTIERUNG

Im Zuge der Entwicklung des Operateursarbeitsplatzes sind eine Anzahl von Prototypen entstanden. Jeder wurde einer Reihe von umfangreichen und breiten Tests unterworfen. Diese wurden mit steigendem Funktionsumfang immer umfassender. Die Tests waren nicht automatisiert, sondern wurden manuell ausgeführt. In jeder Testiteration wurde, soweit es der jeweilige Implementierungsgrad zuließ, das komplette Versuchszenario getestet. Das heisst, jedes Einzelelement des Operateursarbeitsplatzes wurde geprüft. In SAM wurden, wie im späteren Versuch auch, die verschiedenen Strecken nacheinander getrackt. Dabei wurde das spezifizierte Verhalten gegen die Implementierung überprüft. Die Tests führten am Projekt beteiligte Informatiker, Psychologen und im späteren Verlauf auch Versuchspersonen durch. Endbenutzer konnten erst gegen Ende der Arbeiten als Tester einbezogen werden, da in den früheren Phasen keine Versuchspersonen feststanden. Es ist auch zweifelhaft, ob eine frühere Involvierung von Versuchspersonen zielführend gewesen wäre, da insbesondere vor der Vernetzung mit der Trackingkomponente SAM z.T. signifikante Teile der Funktionalität des Operateursarbeitsplatzes fehlten. Durch Evolutionary Prototyping als Vorgehensmodell bei der Softwareentwicklung empfahl sich eine bottom-up Teststrategie.

Der Prototyp wurde inkrementell erweitert und damit konnten auch kleinere Fortschritte getestet werden. Bei der bottom-up Strategie werden die Detailfunktionen vor den Hauptfunktionen getestet. Nach Erstellung des ersten horizontalen Prototypen wurden für jedes Einzelelement des Operateursarbeitsplatzes eine weitestgehend vollständige Umsetzung vorgenommen. Dadurch wurden zuerst einzelne Elemente entsprechend ihrem Implementierungsgrad Tests unterzogen. Nachdem die Vernetzung umgesetzt war, konnte das Zusammenwirken von SAM und dem Operateursarbeitsplatz in Systemtests geprüft werden. Neben den funktionalen Tests, welche die Vollständigkeit und Korrektheit der Implementierung gegenüber der Spezifikation prüfen, wurden auch nicht-funktionale Tests durchgeführt. Hierbei wurde die Funktionsweise des Operateursarbeitsplatzes untersucht. Dafür wurde das Verfahren Heuristische Evaluation verwendet, das in Teil 3.3.2 vorgestellt wurde. Die so gefundenen Mängel sind keine Implementierungsfehler, sondern Mängel der Spezifizierung. Anders als bei den funktionalen Fehlern wird hier nicht nur die Implementierung angepasst, sondern auch die Spezifikation.

5 DISKUSSION

5.1 ARCHITEKTUR

Ein immer wiederkehrendes Problem des Projektes ist die Performance der Softwarekomponenten. Bisher war das meist die Performance von SAM. Optimal ist die Erzeugung eines Logeintrages alle 39 ms. In den aktuellen Versionen von SAM und dem Operateursarbeitsplatz kann das die meiste Zeit erreicht werden. Allerdings gibt es bei der Komponente SAM immer noch Phasen, in denen die Logeinträge deutlich weiter als die angestrebten 30 ms auseinanderliegen. Während des Projektes wurde eine mögliche Lösung für dieses Problem diskutiert. Der Vorschlag sieht vor, die Datenverarbeitung und Speicherung in eine eigene Komponente auszulagern und so die Anzeige, Datenspeicher und Datenverarbeitung stärker zu trennen. Auch die Verlegung der Komponente auf einen eigenen Rechner ist vorstellbar. Hierbei sollte jedoch im Vorfeld untersucht werden, ob die dann notwendige zusätzliche Netzwerkkommunikation die angestrebten 30 ms zwischen den Logeinträgen nachteilig beeinflusst. Als Folge der Auslagerung würden die bisherigen Datenverarbeitungsfunktionalitäten der Komponenten SAM, Operateursarbeitsplatz und Mentale Anstrengung entfallen. Dadurch verschlanken sich die Komponenten. Das hat eine positive Auswirkung auf Erweiterbarkeit und Wartung. Die Funktionen der drei Komponenten reduzieren sich damit auf Darstellung der GUI und gegebenenfalls Netzwerkkommunikation. Joystickeingaben der MWB werden dann nicht mehr in SAM verarbeitet, sondern an den Server gesendet. Dieser führt die notwendigen Berechnungen durch. Dies sind z.B. die Ermittlung der zurückgelegten Strecke, Feststellung von Kollisionen usw. Die Ergebnisse übermittelt der Server dann wieder an die jeweiligen Abnehmer. Das Logging sollte in diesem Szenario konsequenterweise auch von der neuen zentralen Komponente übernommen werden. Diese Aufgabe hat bisher SAM erfüllt. Über die Optimierung der Einzelkomponenten hinaus sollte die Konzentration der Logik in einer zentralen Einheit auch Vorteile beim Weiterentwickeln und Fehlersuche bieten.

5.2 LOGGING

Ein weiterer Hebel, um die Performance zu steigern oder zumindest bei steigender Komplexität zu stabilisieren, ist das Logging. In der Vergangenheit wurden mehrere Möglichkeiten implemen-

tiert und getestet. Die Hauptursachen für die schlechte Performance sind u.a. die große Anzahl an Loggingvariablen und die hohe Frequenz in der Einträge erzeugt werden. Im Moment wird alle 30 ms ein Logeintrag erzeugt.

5.2.1 *Direktes Filelogging*

Beim direkten Filelogging wird in jedem Loggingschritt alle 30 ms der zu speichernde Eintrag direkt in die Datei im Dateisystem geschrieben. Dieses Vorgehen hat sich als zu langsam und damit nicht geeignet erwiesen. Als Lösung für das Problem werden sämtliche Logdaten einer Fahrt in jedem Schritt erst einmal intern in einem Objekt gespeichert. Nach dem Ende einer Fahrt werden alle Einträge in einer Aktion in eine Datei geschrieben. Damit wird dieser Schritt an eine Stelle verschoben, an der ein zeitlich aufwendiger Zugriff auf das Dateisystem zu vernachlässigen ist.

5.2.2 *XML vs. CSV*

Zu Beginn der vorliegenden Arbeit wurden die Loggingdaten in einer XML-Datei gespeichert. Die Möglichkeiten von XML Daten strukturiert zu speichern wurden allerdings nicht genutzt. Die gleichen Daten lassen sich ohne Informationsverlust in einer flachen CSV-Datei ablegen. Gleichzeitig reduziert sich der Speicherbedarf der Logdateien. Davon positiv beeinflusst wurde auch die anschließende Analyse. Das dort eingesetzte JAVA Programm [9] kann eine deutlich kleinere Logdatei verarbeiten.

In seiner aktuellen Ausgestaltung wird der beschriebende Loggingprozess bei Verlängerung der Strecken vermutlich wieder ein Problem für die Performance. Eine Lösung wäre z.B. der Einsatz einer asynchronen Loggingkomponente, ähnlich dem Service Broker vom Microsoft SQL Server. Ihr werden in jedem Schritt die zu loggenden Daten übergeben. Die weitere Verarbeitung der Daten beeinflusst nicht die Verarbeitung der übergebenden Komponente. Diese würde nach Übergabe sofort mit der nächsten Aufgabe fortfahren. Um die weitere Verarbeitung der Loggingeinträge kümmert sich dann die asynchrone Loggingkomponente. Damit wäre die Länge der Strecke kein Perfomance beeinflussender Faktor mehr. Die nächste Grenze wäre beispielsweise eine Größenbegrenzung der Dateigrösse.

5.3 KONFIGURIERBARE GUI

Die Anordnung der einzelnen Elemente des Operateursarbeitsplatzes sind in Form von Koordinaten in der Klasse OPWindowRectangleMorph hinterlegt. Die bisher einzige Möglichkeit,

die Elemente exakt und dauerhaft neu anzuordnen, ist die Anpassung dieser Einträge. Das ist weder intuitiv noch für jeden möglich. Es setzt Wissen über die Squeak Entwicklungsumgebung vorraus. Darüber hinaus wird Wissen über den Speicherort der Koordinaten benötigt. Als Lösung sollte eine zusätzliche GUI Funktionalität geschaffen werden, mit deren Hilfe man die einzelnen Elemente verschieben und die neuen Positionen speichern kann. Denkbar wäre auch noch eine unterstützende Funktion in Form eines Rasters, um eine gleichmässige Ausrichtung mehrerer Elemente zu ermöglichen. Die Möglichkeit, verschiedene Konfigurationen zu speichern und bei Bedarf abzufrufen, sollte in diesem Zusammenhang implementiert werden.

5.4 SKALIERBARKEIT DER GUI

Die aktuelle Implementierung des Operateursarbeitsplatzes wurde auf den verwendeten 30"Bildschirm optimiert. Das umfasst z.B. die Schrift- und Fenstergrösse sowie die Positionierung von Unterelementen wie Buttons oder Schieberegler. Vor allem Grössenangaben sind überwiegend im Programmcode verankert. Die Darstellung auf kleineren Bildschirmen ist dadurch nur eingeschränkt möglich. Einige Elemente sind nur teilweise (z.B. Streckenvorschau) oder gar nicht (z.B. Steuerungsanteil) sichtbar. Die Lösung wäre ein Dialog, mit dessen Hilfe man die wichtigsten Merkmale anpassen kann. Alternativ könnte man den Operateursarbeitsplatz für die Anzeige einiger, ausgewählter Bildschirmauflösungen modifizieren.

5.5 NETZWERKVERHALTEN

Aktuell muss bei der Versuchsvorbereitung die IP Adresse der zu verbindenden Komponenten bekannt sein. Dieses Vorgehen ist umständlich, da der Versuchsleiter die einzelnen IP Adressen ablesen und an die anderen Komponenten übertragen muss. Besser wäre hier eine Funktionalität, welche das eigene Subnetz nach anderen ATEO Komponenten durchsucht. Die mitgelieferten Bibliotheken von Squeak stellen bereits alle notwendigen Funktionen zur Verfügung. Damit reduziert sich der Aufwand um Versuche vorzubereiten. Desweiteren wird die Bedienung intuitiver.

5.6 DATENTYP FÜR PIXEL

In jedem 30 ms Schritt wird die Eingabe der MWB aus den Joysticksensoren ausgelesen und die zurückgelegte Strecke errechnet. Durch die Formel zur Berechnung der zurückgelegten

Strecke ist es möglich, dass dabei Werte mit Nachkommastellen errechnet werden. Die zurückgelegte Strecke, die Schrittweite, wird dann auf die y-Achse der aktuellen Streckenposition addiert und die Strecke bewegt. Die Nachkommastelle der Zahl wird hierbei ignoriert, aber trotzdem gespeichert. Das bedeutet, eine Schrittweite von 10.6 überträgt sich zu einer Streckenbewegung von 10 Pixeln. Wird im nachfolgenden Schritt eine Schrittweite von z.B. 8.5 ermittelt, so wird die Strecke im Ergebnis um 19 Pixel bewegt. In der Logdatei wird die Schrittweite und die insgesamt zurückgelegte Strecke auch mit Nachkommastellen abgelegt. Die Berechnung der zurückgelegten Strecke erfolgt mit Nachkommastellen. In der Darstellung werden diese jedoch nicht berücksichtigt. Das führt zu Irritationen. Daher sollte in den nächsten Versionen versucht werden, bereits bei der Ermittlung der Schrittweite Zahlen vom Typ Integer zu ermitteln. Damit würden sich die folgenden Prozesse automatisch korrigieren.

5.7 ZUSAMMENSTELLUNG DER STRECKE

Auf Basis einer Streckenbeschreibung wird eine Strecke in SAM zur Laufzeit zusammengesetzt. Diese Streckenbeschreibung besteht aus einer Auflistung der Dateinamen der Streckenteile. Die Erstellung neuer oder die Veränderung bestehender Strecken ist sehr aufwändig. Der Benutzer muss die Strecke also erst im Kopf zusammensetzen und danach in Form einer Liste von Dateinamen in der gewünschten Reihenfolge in einer Textdatei ablegen. Eine unterstützende, visuelle Komponente fehlt bei diesem Vorgehen. Eine Lösung für dieses Problem könnte ein Programm bieten, welches eine Art Baukasten zur Streckenerstellung bereitstellt. Dazu gehört eine visuelle Darstellung aller verfügbaren Streckenteile. Der Benutzer sollte die Möglichkeit haben, einzelne Streckenteile auszuwählen und sie zu einer Strecke zusammensetzen zu können. Das Programm leitet aus dem Ergebnis eine Streckendefinition im bereits bekannten Format ab. Das ist die Vorraussetzung, um an keinen Anpassungsbedarf an SAM hervorzurufen. Bereits bestehende Strecken könnten somit ebenfalls visualisiert und verändert werden. Eine weitere mögliche Funktion wäre die Platzierung von Hindernissen auf der Strecke und auch die Ablage einer Hinderniskonfiguration im bekannten Format. Für die Streckenvorschau des Operateursarbeitsplatzes mussten die Strecken bisher immer in einem Bildeditor zusammengesetzt, skaliert und abgelegt werden. Hierdurch erhöht sich zusätzlich der Aufwand von Neuerstellung und Änderungen. Aus diesem Grund sollte das Streckenentwurfsprogramm die Speicherung einer skalierten Version der kompletten Strecke ermöglichen. Durch diese Erweiterung würde die Benutzbarkeit und Zugänglichkeit des Systems ATEO stark profitieren.

6 AUSBLICK

Die im Zuge dieser Arbeit erstellten Softwarekomponenten haben die gestellten Anforderungen sehr gut erfüllt. Positiv hervorzuheben ist die Simulationsfähigkeit des Operateursarbeitsplatzes. Diese war nicht Teil der Anforderungen, sondern ist als Testunterstützung entworfen worden. Es zeigte sich, dass diese Funktion auch gut für Anschauungszwecke eingesetzt werden kann. Dadurch kann der Operateursarbeitsplatz auch ohne verbundene Trackingkomponente verwendet und ein versuchsnahes Verhalten simuliert werden. Sämtliche Buttons und Schieberegler sind bedienbar und geben die entsprechende Rückmeldung innerhalb des Operateursarbeitsplatzes. Die Streckenvorschau, die Darstellung des Inputs der MWB und die visuellen Hinweise sind ebenfalls voll funktionsfähig. Die Simulation kann durch künftige Arbeiten erweitert werden. Zum jetzigen Zeitpunkt wird nur eine Strecke in der Streckenvorschau abgefahren. Es findet kein Wechsel zwischen verschiedenen Strecken statt. Nach Beendigung einer Fahrt beginnt die gleiche Strecke erneut. Die gezeigten simulierten Eingaben der MWB korrespondieren nicht zu den Bewegungen des Trackingobjektes. Auch das Element „Anstrengung“ wird nicht verändert. Dieses Element zeigt die mentale Beanspruchung an der MWB an. Zur Verbesserung der Simulation sollten diese Punkte angepasst werden. Künftige Versionen von Squeak sollten immer wieder auf neue oder verbesserte Funktionen geprüft werden. Im besten Fall lassen sich damit Fehler beheben, bestehende Features verbessern oder neue Operateursarbeitsplatz-Funktionen implementieren. Ein weiteres Arbeitspaket ist das bisher fehlende systematische Testen. Dazu gehören auch klar definierte und wiederholbare Testfälle. Wie SUnit für die GUI eingesetzt werden kann, konnte im Rahmen dieser Arbeit nicht abschließend geklärt werden. Auch wenn neue Anforderungen an das ATEO Testsystem formuliert werden, sollten anschließende Arbeiten auch einen Teil der Ressourcen für die Stabilisierung des Status Quo aufwenden. Das beschränkt sich nicht nur auf den Operateursarbeitsplatz, sondern umfasst alle Teile des ATEO Testsystems. Im Mittelpunkt dieser Bemühungen sollten die Verbesserung der Dokumentierung des Quellcodes und eine fortlaufende Coderevision stehen. Die Steuerzentrale des Operateursarbeitsplatzes bietet ebenfalls Möglichkeiten der Verbesserung. Mit ihrer Hilfe kann der Versuchsleiter den Operateursarbeitsplatz gemäß seinen Wünschen und Anforderungen zusammenstellen. Die Steuerzentrale ist über den Verlauf der

Arbeit entstanden und erweitert worden. Angesichts der inzwischen gewachsenen Funktionsumfanges wäre eine Überarbeitung dieses GUI-Elementes angebracht. In seiner derzeitigen Form ist es nur aufwendig um neue Funktionen erweiterbar.

7

ZUSAMMENFASSUNG

Mit der vorliegenden Arbeit wurde das ATEO Testsystem um den Operateursarbeitsplatz erweitert. Die wichtigsten Vorarbeiten sind die Diplomarbeiten von Hermann Schwarz [19] und Nicolas Niestroj [18]. In seiner Diplomarbeit schaffte Hermann Schwarz die Vorraussetzungen für das User Interface des Operateursarbeitsplatzes. Er erarbeitete u.a. Gestaltungsrichtlinien. Als ein Ergebnis seiner Arbeit erstellte er eine erste Version des Operateursarbeitsplatzes. Nicolas Niestroj leistete mit den Ergebnissen aus seiner Arbeit einen wesentlichen Beitrag für die Implementierung der Netzwerkkommunikation. Die von ihm erarbeiteten Implementierungen der Kommunikation via Sockets in Squeak waren ein wesentlicher Einfluss für die aktuelle Umsetzung. Die vorangehende Studienarbeit [12] bildet ebenfalls eine wichtige Grundlage. Dort wurde unter anderem mit Möglichkeiten der Netzwerkkommunikation experimentiert. Die Machbarkeit und der Aufwand eines Updates auf eine neuere Squeak Version wurde ebenso untersucht. Das Ziel der Arbeit war es, das ATEO Testsystem um den Operateursarbeitsplatz zu erweitern, um damit Versuche durchführen zu können. Dies wurde erreicht. Dazu wurde ein User Interface nach Vorgaben der Fachseite erstellt. Die Verarbeitungslogik der Operateurseingaben wurde implementiert, um ein Zusammenwirken der Trackingskomponente SAM und dem Operateursarbeitsplatz zu ermöglichen. Bei der Entwicklung der Software wurde Evolutionary Prototyping als Vorgehensmodell eingesetzt. Das ermöglichte eine frühe Erstellung von Prototypen im Prozess. Dies schuf die Vorraussetzung einer besseren Abstimmung zwischen Psychologen und dem Umsetzer. Damit konnte der Prototyp als Grundlage für das weitere Vorgehen dienen. Das Vorgehensmodell wurde durch die im „Manifest für Agile Softwareentwicklung“ formulierten Prinzipien ergänzt. Das Ziel agiler Softwareentwicklung ist es, Bürokratie und Regeln auf ein erforderliches Mindestmaß zu reduzieren. Während das Vorgehensmodell die einzelnen Schritte im Prozess bestimmt, hatten die agilen Prinzipien großen Einfluss darauf, wie die einzelnen Schritte gestaltet wurden. Die gesamte Entwicklung des Operateursarbeitsplatzes war ein sehr dynamischer Prozess, d.h. die Spezifikationen durchliefen eine Reihe von Änderungen. Das wirkte sich meist auch auf bereits implementierte Teile aus. Über die gesamte Dauer der Arbeit wurde eine enge Zusammenarbeit mit Psychologen und anderen am Projekt beteiligten Informatikern gepflegt. Am Ende der vorliegenden Arbeit

steht eine stabile Version des Operateursarbeitsplatzes, die sich in nahezu 80 Versuchen bewährt hat.

Teil I

ANHANG

A

ANHANG

A.1 VERSIONEN DES OPERATEURSARBEITSPLATZES

A.1.1 *Version 1.1.7*

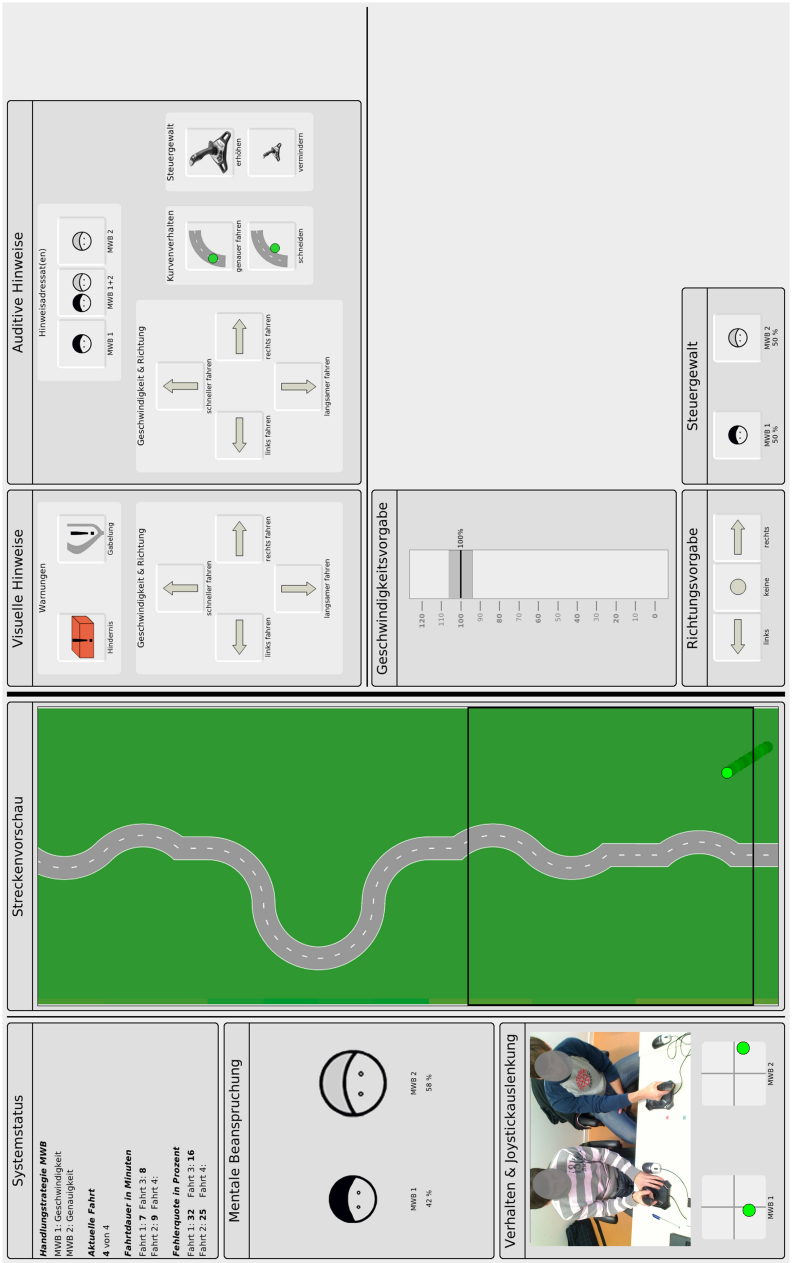

Abbildung 38: Operateursarbeitsplatz Version 1.1.7

A.1.2 *Version 2.2.0*

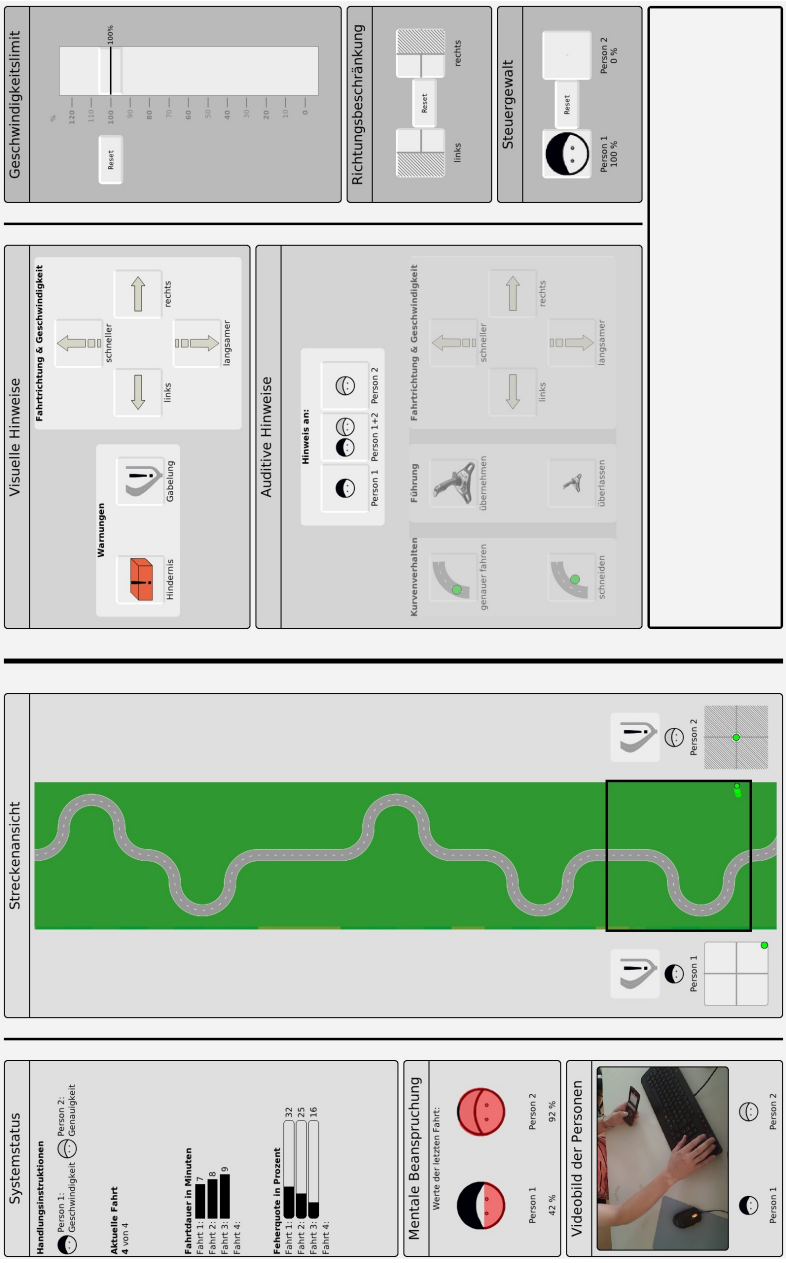

Abbildung 39: Operateursarbeitsplatz Version 2.2.0

A.1.3 *Version 2.4.0*

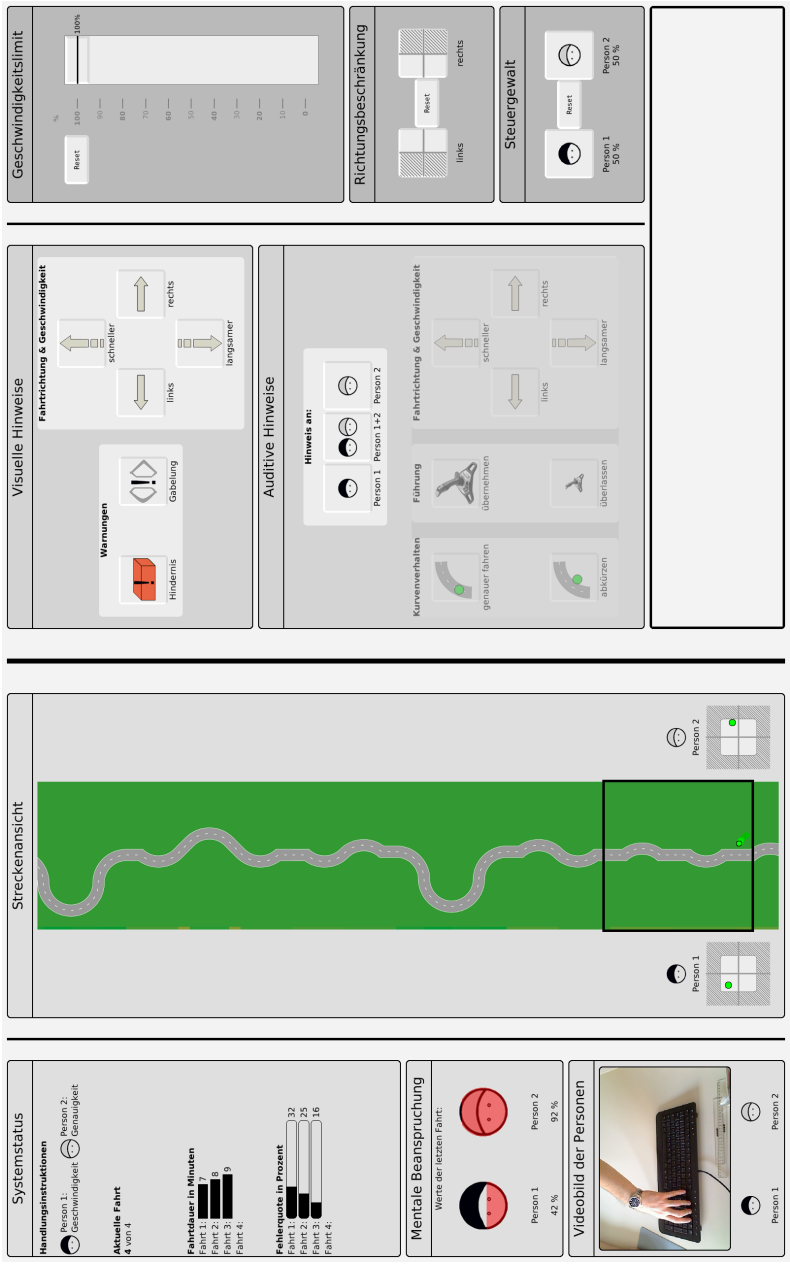

Abbildung 40: Operateursarbeitsplatz Version 2.4.0

A.1.4 *Version 2.6.0*

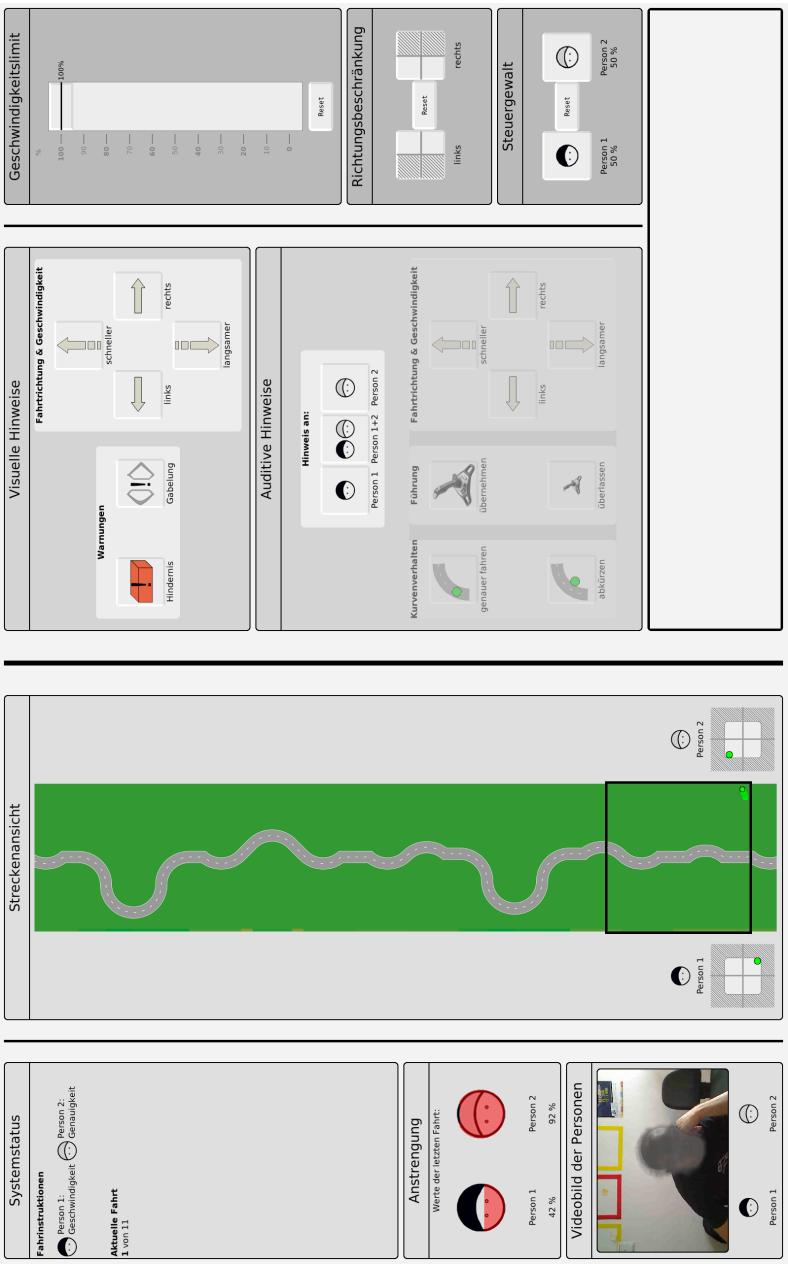

Abbildung 41: Operateursarbeitsplatz Version 2.6.0

A.1.5 *Version 2.7.0*

Abbildung 42: Operateursarbeitsplatz Version 2.7.0

A.1.6 *Version 2.7.1*

Abbildung 43: Operateursarbeitsplatz Version 2.7.1

A.1.7 *Version 2.8.0*

Abbildung 44: Operateursarbeitsplatz Version 2.8.0

A.1.8 *Version 2.8.4*

Dies ist die finale Version die im Rahmen dieser Diplomarbeit erstellt und komplett getestet wurde.

Abbildung 45: Operateursarbeitsplatz Version 2.8.4

A.2 VERSIONEN DER VORLAGEN

A.2.1 *Vorlage Version 1.1*

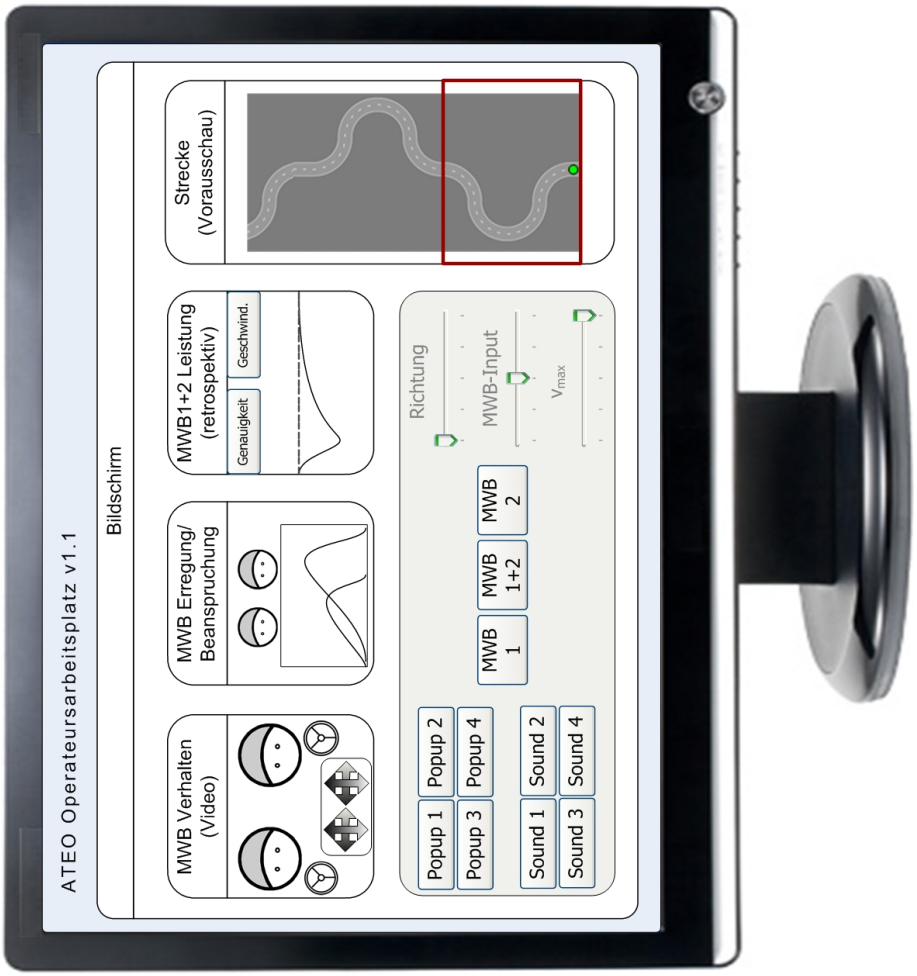

Abbildung 46: Vorlage Version 1.1

A.2.2 *Vorlage Version 1.3*

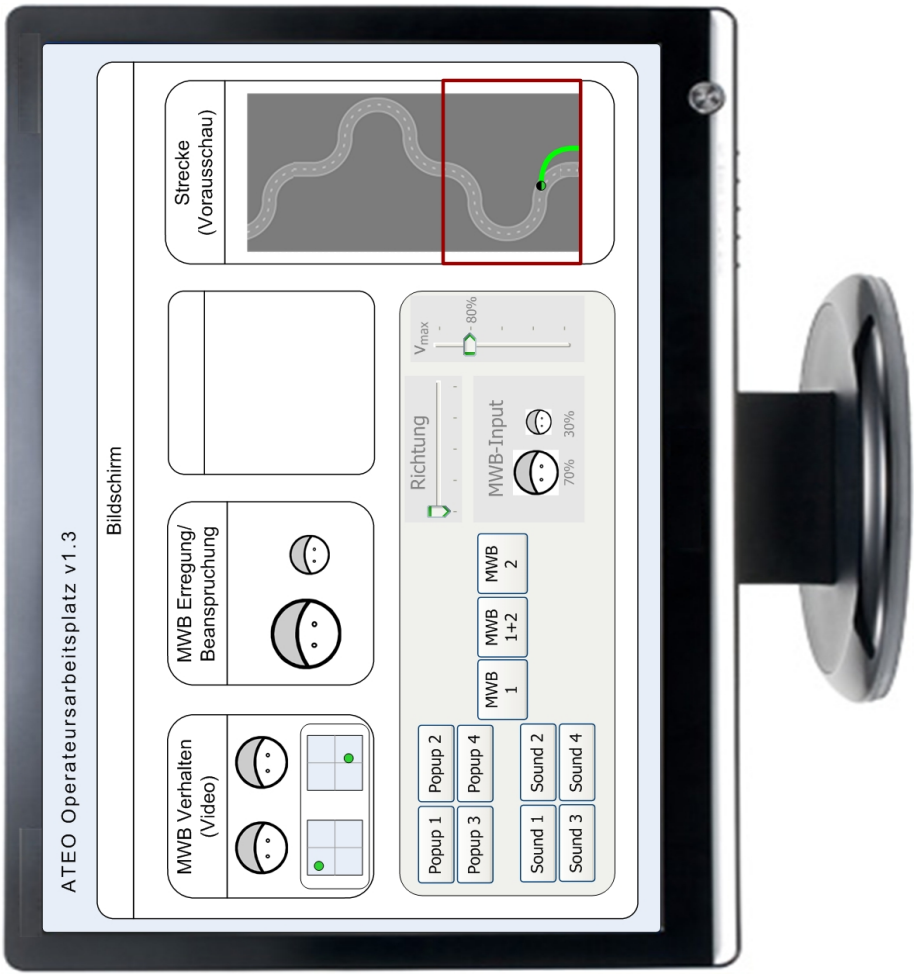

Abbildung 47: Vorlage Version 1.3

A.2.3 *Vorlage Version 1.8*

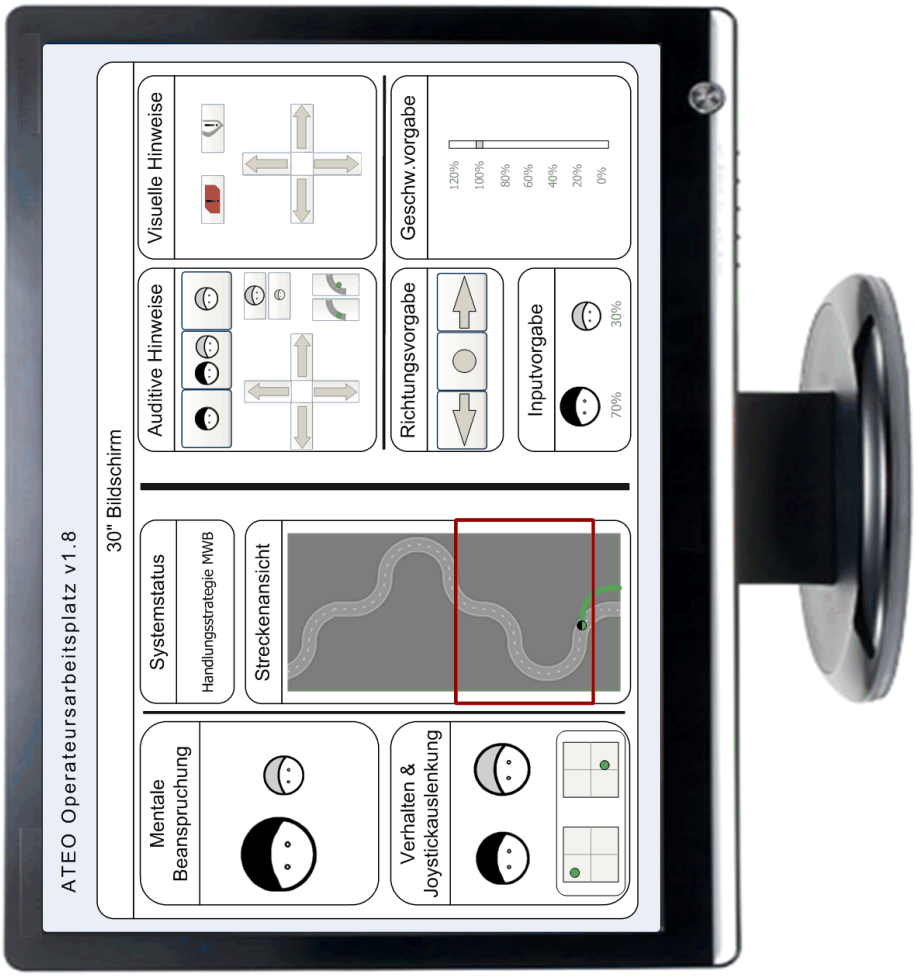

Abbildung 48: Vorlage Version 1.8

A.2.4 *Vorlage Version 2.0*

Abbildung 49: Vorlage Version 2.0

A.2.5 *Vorlage Version 2.4*

Abbildung 50: Vorlage Version 2.4

A.3 BEISPIELE AUS DEM QUELLCODE

A.3.1 *Singletonimplementierung*

Im Codelisting wird die Singletonimplementierung am Beispiel der modelData-Klasse gezeigt.

Listing 2: Implementierung der Klassenmethode „getInstance“

```
getInstance

        (Current isNil)
        ifTrue: [ Current := self basicNew
           initialize. ].

        ^Current
```

A.3.2 *Implementierung des Steppings*

Um die Steppingfuktionalität eines Morph nutzen zu können, muss die Methode „stepTime“ implementiert werden. Dort wird die Zeit zwischen den einzelnen Schritten festgelegt.

Listing 3: Implementierung der Methode „stepTime“

```
stepTime

        ^100
```

Zusätzlich muss neben der Methode „stepTime“ auch noch die Methode „step“ implementiert werden. Sie enthält die Instruktionen, die während eines Steps abgearbeitet werden.

Listing 4: Implementierung der Methode „step“

```
step

        [....]
        someStringData do: [:token |    self
           process: token]
        [....]
```

Mit dem Aufruf „self startStepping“ am Morph wird das Stepping begonnen. Im Takt der in der Methode „stepTime“ angegebenen Zeit werden die unter „step“ hinterlegten Funktionen abgearbeitet.

A.4 PRINZIPIEN HINTER DEM MANIFEST FÜR AGILE SOFTWAREENTWICKLUNG

1. Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.
2. Heisse Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
3. Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
4. Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.
5. Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.
6. Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteam zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.
7. Funktionierende Software ist das wichtigste Fortschrittsmaß.
8. Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
9. Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.
10. Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell.
11. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
12. In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

A.5 UML KLASSENDIAGRAMM DES OPERATEURSARBEITSPLATZES

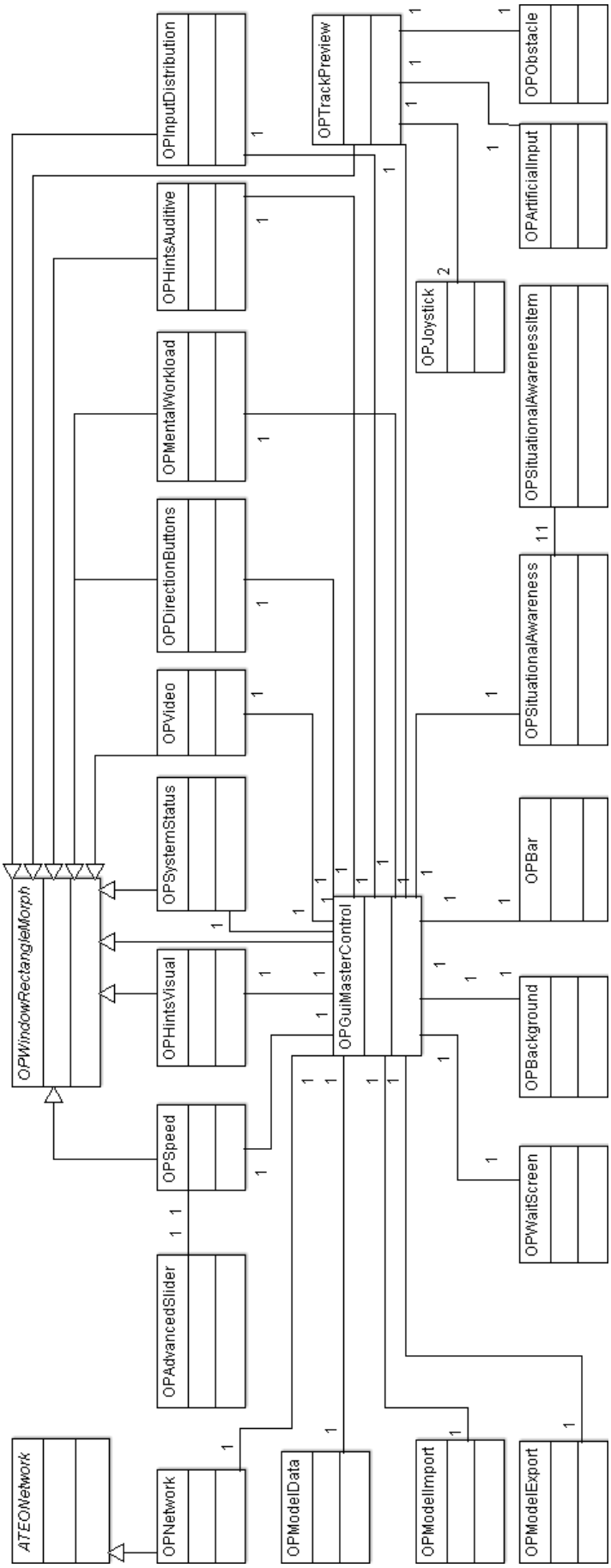

Abbildung 51: Klassendiagramm der Version 2.8.4 des Operateursarbeitsplatzes

LITERATURVERZEICHNIS

[1] H. Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Spektrum Lehrbücher der Informatik. Spektrum Akademischer Verlag, Heidelberg, 2011.

[2] H. Balzert, R. Koschke, U. Lämmel, P. Liggesmeyer, J. Quante, and H. Balzert. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum Lehrbücher der Informatik. Spektrum Akademischer Verlag, Heidelberg, 2009.

[3] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. Website, 2001. Available online at `http://http://www.agilemanifesto.org`; visited on January 5th 2013.

[4] Klaus Bothe, Michael Hildebrandt, and Nicolas Niestroj. *ATEO-SYSTEM KOMPONENTE: SAMS - SOFTWARE-SPEZIFIKATION*. Humboldt Universität, Berlin, 2010.

[5] Maik Burandt. *ATEO - begleitende Arbeit zur Projektdurchführung*. Studienarbeit, Berlin, 2007.

[6] Mica R. Endsley. Design and evaluation for situation awareness enhancement. *Proceedings of the Human Factors Society 30th Annual Meeting*, pages 97–101, 1988.

[7] Nina Gérard, Stefanie Huber, Jens Nachtwei, Udo Schubert, and Bram Satriadarma. *A Framework for Designers to Support Prospective Design*. Centre of Human-Machine Systems Graduateschool prometei: Prospective Design of Human-Machine Interaction Technische Universität Berlin, Berlin, 2010.

[8] Barbara Gross and Jens Nachtwei. *How to develop and use assistance systems efficiently - Using the microworld to acquire knowledge for developers and operators*. Centre of Human-Machine Systems Graduateschool prometei: Prospective Design of Human-Machine Interaction Technische Universität Berlin, Berlin, 2007.

[9] Tobias Hampel. *Weiterentwicklung und Verbesserung einer Analysesoftware für dynamische Versuchsumgebungen*. Diplomarbeit, Berlin, 2012.

[10] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: the story of squeak, a practical smalltalk written in itself. *OOPSLA 97: Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 318–326, 1997.

[11] Edward Klimas, Suzanne Skublics, and David A. Thomas. *Smalltalk With Style (Volume One)*. Prentice Hall, 1996.

[12] Christian Leonhard. *Fenster zum Prozess: Weiterentwicklung eines Operateursarbeitsplatzes im Projekt Arbeitsteilung Entwickler Operateur (ATEO)*. Studienarbeit, Berlin, 2010.

[13] Jens Nachtwei. The many faces of human operators in process control: a framework of analogies. *Theoretical Issues in Ergonomics Science*, 2010.

[14] Jens Nachtwei. A multi-level design approach for a supervisory control master display in human factors experiments. *Behaviour And Information Technology*, 2010.

[15] Jens Nachtwei. *Design and Evaluation of a Supervisory Control Lab System for Automation Research - A Theoretical and Empirical Contribution to the Discussion on Function Allocation*. Dissertation, Humboldt Universität zu Berlin, 2011.

[16] Jakob Nielsen and Robert L. Mack. *Usability Inspection Methods*. John Wiley & Sons, Inc., NY, 1994.

[17] Nicolas Niestroj. *Erweiterung des ATEO-Systems zur Komplexitätserhöhung von SAM*. Studienarbeit, Berlin, 2008.

[18] Nicolas Niestroj. *Vernetzung im ATEO Projekt aus inhaltlicher und technischer Sicht*. Diplomarbeit, Berlin, 2009.

[19] Hermann Schwarz. *Fenster zum Prozess: ein Operateursarbeitsplatz zur Überwachung und Kontrolle von kooperativem Tracking*. Diplomarbeit, Berlin, 2009.

[20] J. Seppälä, A. S. Nissinen, H. Koivo, and M Laitila. Intelligent visualization of dynamic process data. In *Control Systems 2000 - Quantifying the Benefits of Process Control - Preprint*, pages 85–88, Canada, 2000. PAPTAC, STFI, Finnish Paper Engineers Association. Control Systems 2000, Victoria BC, Canada, May 1-4, 2000.

[21] 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, 2005.

[22] K.J. Vicente, R.J. Mumaw, and E.M. Roth. Operator monitoring in a complex dynamic work environment: a qualitative cognitive model based on field observations. *Theoretical Issues in Ergonomics Science, 5 (5)*, pages 359–384, 2004.

[23] H. Wandke and J. Nachtwei. The different human factor in automation: the developer behind vs. the operator in action. *D. de Waard, F.O. Flemisch, B. Lorenz, H. Oberheid, and K.A. Brookhuis (Eds.), Human factors for assistance and automation*, pages 493–502, 2008.

SELBSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Diplomarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 21. Januar 2013 .....