Die Integration von Automatiken in das ATEO-Masterdisplay (AMD)

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

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

eingereicht von:	Stefan Schulze
geboren am:	26. Oktober 1984
in:	Berlin
Gutachter:	Prof. Dr. Klaus Bothe
	Prof. Dr. Hartmut Wandke
eingereicht am:	.....
verteidigt am:	.....

Abstract

This diploma thesis is part of the project ATEO (Arbeitsteilung Entwickler-Operateur, division of work between developers and operators). ATEO is the description of a project of the Graduiertenkolleg prometei (Prospektive Gestaltung von Mensch-Technik-Interaktion) from the Technische Universität Berlin in combination with the institute of psychology from the Humboldt Universität zu Berlin.

The intention of the project ATEO is not to explore the funcional seperation between human and automatic systems with comparing the performance from an operator and an automatic system in the traditional way. The project is rather aimed at the comparision from the performance of an operator and the developer of the automatic systems.

The central part of the project is the Socially Augmented Microworld (SAM), where two test persons (the socalled Microworld Inhabitants (MWI)) are steering an virtual object along a track. During this tracking task, the track is partially armed with different hints like obstacles, curves or forks. This complex dynamic process is supervised and optimized by a third test person (the operator) at the ATEO-Masterdisplay (AMD). In summary the test environment is similar to a controlling unit e.g. from a power station or a control tower from an airport.
Automatic systems, which are designed by developers, are providing another opportunity to supervise and optimize the tracking process.

The result of this diploma thesis is a third cooperative test mode between the automatic system and the operator, who can activate and deactivate the automatic systems by the AMD. Therefore it was imperative to adapt the analysis tool to the informations that are resulted by the new mode.

Zusammenfassung

Diese Diplomarbeit ist dem ATEO-Projekt zugeordnet und angesiedelt am Lehrstuhl für Ingenieurpsychologie/Kognitive Ergonomie des Instituts für Psychologie der Humboldt-Universität zu Berlin. Arbeitsteilung Entwickler-Operateur (ATEO) ist die Bezeichnung eines Projekts im Rahmen des Graduiertenkolleg prometei (Prospektive Gestaltung von Mensch-Technik-Interaktion) der Technischen Universität Berlin.

Ziel des ATEO-Projekts ist es, die Funktionsteilung zwischen Mensch und Automatik zu erforschen, wobei entgegen traditioneller Ansätze nicht die Leistung von Operateur und Automatik, sondern die Leistung von Operateur und Entwickler (von Automatiken) verglichen wird.

Die zentrale Komponente des Projekts ist die Socially Augmented Microworld (SAM), in der zwei Versuchspersonen (sogenannte Mikroweltbewohner (MWB)) gemeinsam ein virtuelles Objekt eine Strecke entlang steuern sollen. Während dieser Trackingaufgabe werden Konfliktsituationen durch Hindernisse, Kurven und Gabelungen auf der Strecke erzeugt. Dieser komplexe dynamische Prozess wird durch eine dritte Versuchsperson (Operateur) am ATEO-Masterdisplay (AMD) überwacht und optimiert. Die Versuchsumgebung ähnelt der eines Überwachungssystems, z.B. der Leitwarte in einem Kraftwerk oder der Flugüberwachung durch Lotsen.
Anstelle des Operateurs ist es ebenso möglich, den Trackingprozess durch eine von Entwicklern konzipierte Automatik zu überwachen und zu optimieren.

Als Ergebnis dieser Diplomarbeit ist ein dritter kooperativer Modus zwischen Automatik und Operateur entstanden, in dem dieser am AMD bei Bedarf Automatiken aktivieren und wieder deaktivieren kann. Die Aufzeichnung der Daten und deren Aufbereitung wurden entsprechend angepasst.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Akronyme

AAF	ATEO-Automation-Framework
ATEO	Arbeitsteilung Entwickler-Operateur
ALS	ATEO-Lab-System
AMD	ATEO-Masterdisplay
CSV	Dateiformat für einfach strukturierte Daten
GUI	Graphical User Interface (grafische Benutzeroberfläche)
LFA	Logfile-Analysetool
MWB	Mikroweltbewohner
MWI	Microworld Inhabitants (Mikroweltbewohner)
OA	Operateursarbeitsplatz
SAM	Socially Augmented Microworld
XML	Extensible Markup Language

1 Einleitung

Dieses erste Kapitel stellt eine Einleitung zur Diplomarbeit dar. Der Inhalt besteht im Wesentlichen aus einer Einbettung in das Projekt und der Motivation, dieses zu erweitern. Darauf aufbauend folgt das Ziel und die Gliederung dieser Diplomarbeit.

1.1 Einbettung

Arbeitsteilung Entwickler-Operateur (ATEO) ist die Bezeichnung eines Projekts im Rahmen des Graduiertenkolleg prometei (Prospektive Gestaltung von Mensch-Technik-Interaktion) der Technischen Universität Berlin und angesiedelt am Lehrstuhl für Ingenieurpsychologie/Kognitive Ergonomie vom Institut für Psychologie der Humboldt-Universität zu Berlin.

Ziel des ATEO-Projekts ist es, die Funktionsteilung zwischen Operateuren (Benutzern) und Entwicklern von Automatiken in Hinsicht auf die Leistung bei der Überwachung und Regulierung eines komplexen dynamischen Prozesses zu erforschen. Im ATEO-Versuchsablauf überwacht eine Versuchsperson (der Operateur) diesen Prozess, in welchem zwei weitere Probanden (sogenannte Mikroweltbewohner (MWB)) eine Tracking-, Manöver- und Navigationsaufgabe absolvieren. Dabei steuern sie gemeinsam ein virtuelles Objekt eine Versuchsstrecke entlang. Die MWB, die Versuchsstrecke, das virtuelle Objekt sowie Technikkomponenten wie Joysticks und ein Monitor sind Teile des Subsystems der Socially Augmented Microworld (SAM).
Um die Komplexität des Prozesses zusätzlich zu erhöhen, werden Konfliktsituationen[1] erzeugt. Die Aufgabe des Operateurs besteht darin, die Trackingleistung zu optimieren, z.B. indem er standardisierte auditive Hinweise an einen oder beide MWB oder visuelle Hinweise an beide gibt. So kann der Operateur die MWB im Vorhinein auf Konfliktsituationen aufmerksam

[1] z.B. Hindernisse, Kurven und Gabelungen

machen oder die Bewältigung solcher Situationen im Nachhinein loben. Dem Operateur ist ebenso die Möglichkeit gegeben, die Geschwindigkeit der MWB zu begrenzen, die Fahrtrichtung nach links oder rechts zu blockieren oder die Verteilung der Joystick-Inputs zu manipulieren. Die Versuchsumgebung ähnelt der eines Überwachungssystems, z.B. der Leitwarte in einem Kraftwerk oder der Flugüberwachung durch Lotsen.

Bisherige Studien untersuchten einerseits die Leistung des Operateurs (Effizienz, Effektivität und Sicherheit). Im Zuge der Weiterentwicklung des ATEO-Systems entstanden andererseits Automatiken durch Software-Entwickler, welche die MWB ohne den Einfluss eines Operateurs überwachen und unterstützen. Die Leistung der Entwickler-Automatiken wird dann der des Operateurs vergleichend gegenübergestellt.

Anders als bei Operateuren können die Entwickler der Automatiken nicht spontan in das System eingreifen. Sie können Situationen analysieren und antizipieren, um dann wohlüberlegte Entscheidungen über die Art der Unterstützung der MWB durch eine Automatik zu treffen in der Hoffnung, keine der möglichen Situationen unbehandelt gelassen zu haben. Die Operateure müssen in Entscheidungssituationen spontan reagieren, haben aber den Vorteil, im gleichen Moment die Ergebnisse zu sehen.

1.2 Motivation

Beim derzeitigen Entwicklungsstand sind also zwei Arten von Assistenzen wählbar:

Im *Operateur-Modus* werden die Arbeitsplätze der MWB und des Operateurs so zusammengeführt, dass die MWB während des dynamischen Tracking-Prozesses von einem Operateur mit verschiedenen Hinweisen unterstützt werden können.

Im *Agenten-Modus* kann diese Unterstützung durch verschiedene Agenten einer Automatik geleistet werden. Diese Agenten können in einem Automatiken-GUI ausgewählt und konfiguriert werden, wobei es möglich ist, dass mehrere Agenten den Prozess und damit das Verhalten der MWB beeinflussen - Agenten führen also die Automatik-Funktionen aus.

Diese Arbeit muss der Operateur ebenso leisten. So muss er in bestimmten Streckenabschnitten mehrere Hinweise sequenziell und mit geringer Latenzzeit geben. Ein Beispiel hierfür ist eine Hindernis-Gabelung-Kombination, die der Operateur erst erkennen muss, um dann rechtzeitig auditive oder visuelle Hinweise in korrekter Abfolge zu geben. Dies kann zu einer hohen Belastung des Operateurs führen.

1.3 Ziel der Diplomarbeit

Ziel der Diplomarbeit ist es, dem Operateur in einem dritten Modus die Benutzung von Automatik-Funktionen zu ermöglichen. Dieser neue Testmodus wird als *Kombination* der bisherigen Modi gesehen, da er die Überwachung des komplexen Trackingvorgangs durch den Operateur vorsieht, welcher bei Bedarf Agenten einer Automatik hinzuzieht. Diese sollen dann bestimmte Teile seiner Aufgaben übernehmen.

Hierfür müssen die Bedienelemente des AMD so überarbeitet werden, dass es dem Operateur möglich ist, bei dessen Betätigung Agenten zu aktivieren, die dann die Aufgaben übernehmen, die er selbst mit Drücken der Schaltfläche abarbeitet. Im Zuge dieser Anforderung müssen mehrere Interfaces entwickelt und die Netzwerkschnittstelle überarbeitet werden, um die einzelnen Software-Komponenten zu verbinden. Ebenso ist es notwendig das SAM-GUI, welches die Versuchsvoreinstellungen verwaltet, so anzugleichen, dass ein neuer Modus anwählbar ist. Im nächsten Schritt sind die Aktivitäten der AMD-Schaltflächen genauso zu loggen wie die der Agenten, welche auf diese Weise angesteuert werden. Am Ende eines jeden Testdurchlaufs entste-

hen Testdaten (Logfiles), die aufbereitet werden müssen. Dafür ist es notwendig, das bestehende Analysetool an die Informationen des neuen Modus anzupassen.

1.4 Aufbau der Arbeit

Das *erste Kapitel* stellt die Einführung in die Diplomarbeit dar. Dem Leser wird nach der Einbettung ins ATEO-System und der Motivation zur Weiterentwicklung das Ziel dieser Diplomarbeit näher gebracht. Die anschließende Gliederung entschlüsselt den Aufbau der schriftlichen und praktischen Arbeit.

Im *zweiten Kapitel* werden die technischen Grundlagen dieser Arbeit definiert. Neben der Einführung in die einzelnen ATEO-Komponenten wird hier der Zusammenschluss als Gesamtsystem beschrieben und auf das Werkzeug zur späteren Auswertung eingegangen. Die Vorstellung der unterschiedlichen Beschreibungs- und Programmiersprachen beinhaltet der letzte Teil dieses Kapitels.

Den theoretischen Teil spiegelt das *dritte Kapitel* wieder. In dieser Diplomarbeit sollen die Erweiterungen des ATEO-Systems als Implementierungen in Frameworks gesehen werden. Dazu werden die zur Erfüllung der Aufgaben benötigten Komponenten als Frameworks identifiziert, eine Auswahl der benötigten Frameworks getroffen und die Vorgehensweise der Eingriffe erläutert.

Im *vierten Kapitel* werden die praktischen Eingriffe in die Module SAM, AAF und AMD erläutert. Dazu wurden vorher die Anforderungen in Muss-, Wunsch- und Abgrenzungskriterien unterteilt und aufgelistet. Der letzte Teil des Kapitels beschreibt die Erweiterung des LFA und die umfangreichen Tests, die durchgeführt wurden.

„Zusammenfassung und Ausblick“ ist das *fünfte Kapitel*, in dem zuerst die Erfolge der Arbeit zusammengefasst werden. Im Anschluss wird über die Probleme bei der Umsetzung der Aufgaben berichtet sowie Ideen und ein Ausblick für den zukünftigen Verlauf und die Weiterentwicklung des ATEO-Projekts vorgestellt.

Im *Anhang* sind die Klassendiagramme der beiden Netzwerkklassen zu finden, die im Zuge dieser Arbeit neu erstellt werden konnten. Sie geben einen Überblick über den Umfang und die Komplexität der Arbeiten. Nach der Aufschlüsselung des Inhalts der beiliegenden DVD folgt als letzter Anhang eine Bedienungsanleitung für das richtige Starten und Benutzen des kooperativen Modus.

2 Technische Grundlagen

In diesem Kapitel wird eine Einführung in die einzelnen Bestandteile des ATEO-Systems gegeben. Neben allen Komponenten, die für Testversuche notwendig sind, können hier auch Informationen über das Zusammenwirken dieser Komponenten im Gesamtsystem und über das Werkzeug zur Auswertung der Testdaten gefunden werden. Im Anschluss werden die angewandten Programmier- und Beschreibungssprachen vorgestellt.

2.1 ATEO-Komponenten

2.1.1 Socially Augmented Microworld (SAM)

Nach Papert und Minsky[8] ist eine Mikrowelt ein schematisches Model einer interaktiven Lernumgebung, indem Dinge wesentlich einfacher als in der realen Welt dargestellt sind. In der SAM, einer um eine menschliche Komponente angereicherte (oder ergänzte) Mikrowelt, agieren zwei Versuchspersonen (MWB) sowie ein virtuell gesteuertes Fahrobjekt. SAM wird demnach durch eine Tracking-, Manöver und Navigationsaufgabe repräsentiert, bei dem die MWB das Objekt entlang einer Versuchsstrecke steuern, für das sie sowohl die Richtung, als auch die Geschwindigkeit aktiv beeinflussen können.

Jeder der MWB bedient dafür einen Joystick, wobei der Input zu jeweils 50% verrechnet wird. Das gemeinsame Ziel ist, das Tracking-Objekt (siehe Abbildung 1) so schnell und so genau wie möglich ins Ziel zu steuern. Dabei erhalten die MWB jedoch unwissentlich zwei gegensätzliche Instruktionen als sekundäres Ziel: so soll einer der MWB möglichst schnell, der andere möglichst genau steuern. Nach jedem Versuchsdurchlauf[2] müssen die MWB auf einer Skala den Grad der jeweils subjektiv empfundenen Anstren-

[2] insgesamt müssen 11 unterschiedliche Strecken befahren werden

gung angeben. Dieser wird anschließend dem Operateur durch das Netzwerk übermittelt und im Statusanzeigen-Bereich seines Displays angezeigt (siehe Abbildung 2, links).

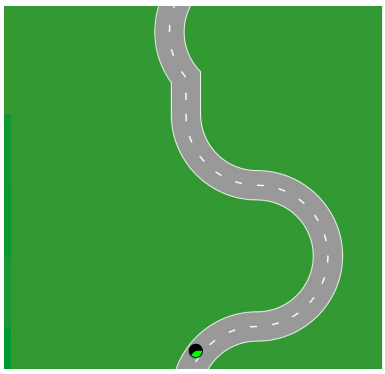


Abbildung 1: *SAM: Tracking-Objekt (unten-mittig) und Versuchsstrecke*

Der dynamische Prozess der SAM darf nicht deterministisch sein, da sonst eine Automatik alle Eingriffe exakt im Voraus berechnen könnte, allerdings aber auch nicht absolut zufällig ablaufen. In diesem Fall wären überhaupt keine sinnvollen Korrekturen durch eine Automatik mehr möglich, da der Entwickler Ereignisse nicht im Vorfeld antizipieren könnte.[3]
Aus diesem Grund wurde die Mikrowelt um Menschen als Versuchspersonen ergänzt. Die SAM ist also ein komplexes System, welches sich realitätsnah verhält und so ein technisches System darstellt, das einigermaßen vorhersehbar ist und unerwartetes Verhalten (z.B. durch Störungen) aufweisen kann.

2.1.2 ATEO-Masterdisplay (AMD)

Das AMD ist das Interface für den Operateur und bietet ihm die Möglichkeit, den laufenden Prozess (SAM) zu überwachen und bei Bedarf einzugreifen (siehe Abbildung 2). Die Eingriffe sind hierbei in harte und weiche Eingriffe zu differenzieren.

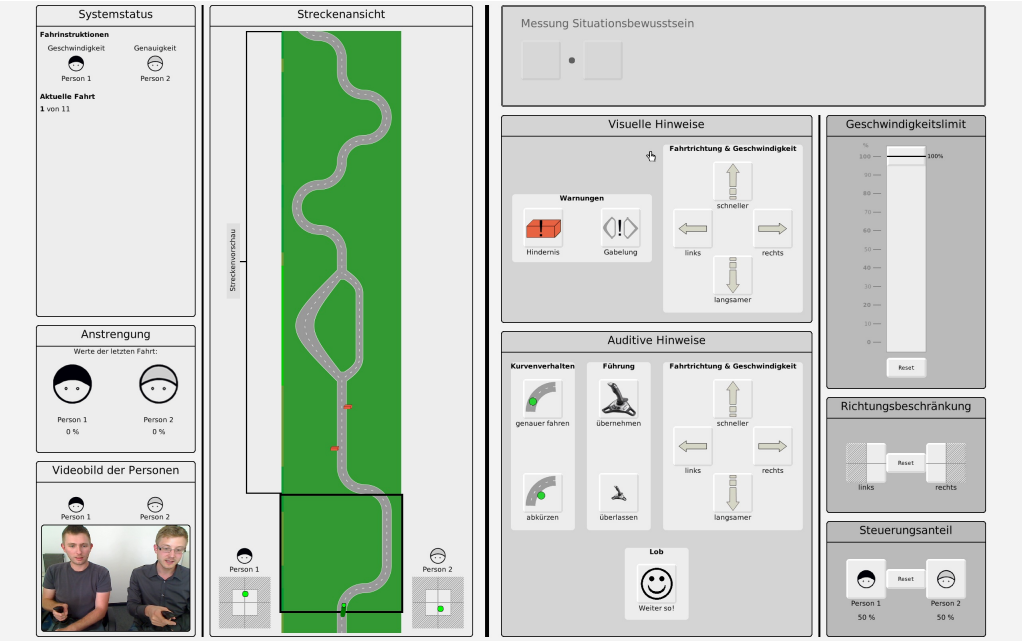
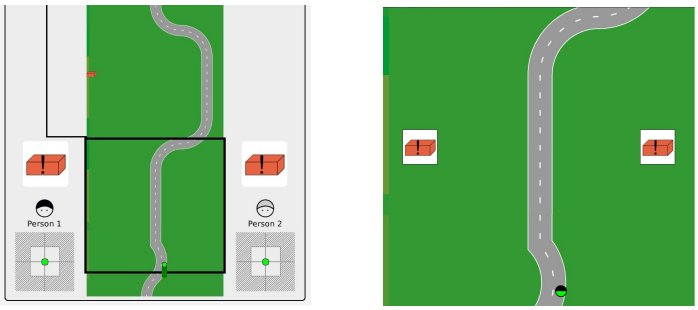


Abbildung 2: *AMD mit Statusanzeigen und Live-Bild der Testpersonen (links), Streckenansicht (links-mittig), weichen Eingriffen (rechts-mittig) und harten Eingriffen (rechts)*

Dem Operateur werden verschiedene Möglichkeiten zur Überwachung der MWB geboten: Im linken Teil des AMDs kann er die MWB durch ein Live-Videobild beobachten und erhält eine grafische Darstellung ihrer Joystickauslenkungen und eine erweiterte Streckenansicht. Diese stellt nicht nur den für die MWB sichtbaren Streckenbereich dar, sondern auch einen zusätzlichen Teil der als nächstes zu befahrenen Strecke (siehe Abbildung 2, Statusanzeigen und Streckenansicht). Dadurch wird dem Operateur ermöglicht,

Konfliktsituationen zu antizipieren und sich auf die Unterstützung der MWB vorzubereiten, wenn diese sich Hindernissen oder Gabelungen nähern. Auch zeigt die Streckenansicht an, wie sich die Mikroweltbewohner in Bezug auf Genauigkeit und Geschwindigkeit in der Vergangenheit verhalten haben. Dies wird durch einen Schweif verdeutlicht, der seine Farbe gemäß der gefahrenen Geschwindigkeit anpasst. Ein kurzer Schweif zeigt ein sich langsam und ein langer Schweif ein sich schnell bewegendes Tracking-Objekt an. Detaillierte Informationen dazu können in der Diplomarbeit von S. Schwarz[9] nachgelesen werden.

Als weiche Eingriffe sind dem Operateur auditive und visuelle Hinweise zur Hand gegeben, um damit den steuernden MWBn Hilfestellungen in Hinblick auf ihr gemeinsames Ziel (die schnelle und fehlerfreie Bewältigung der Strecke) zu geben. Die möglichen visuellen Eingriffe werden funktionell in zwei Klassen unterteilt. Zum einen in Warnungen, welcher die Buttons „Hindernis“ und „Gabelung“ zugewiesen sind, wobei die Hindernisse statisch auf einem Streckenbereich verteilt sein können oder dynamisch von links nach rechts die Strecke kreuzen. Gabelungen spalten den Weg in einen linken und rechten Abschnitt auf. Zum anderen wird in die Klasse „Fahrtrichtung und Geschwindigkeit“ unterteilt. Hierbei handelt es sich um Buttons mit verschiedenen Pfeilmotiven, die nach der Anwahl am AMD den MWBn anzeigen, dass sie schneller, langsamer, links oder rechts fahren sollen. Drückt der Operateur einen der sechs Schaltflächen, so wird ein visueller Hinweis für 1500ms im Bereich der Streckenvorschau im AMD angezeigt sowie mittig-links und mittig-rechts in der SAM (siehe Abbildung 3).



(a) AMD (b) SAM

Abbildung 3: *AMD und SAM: Anzeige des visuellen Hinweises „Hinderniswarnung"; links: AMD, rechts: SAM*

Auditive Hinweise sind, anders als visuelle, in vier Bereiche unterteilt: Kurvenverhalten, Führung, Fahrtrichtung und Geschwindigkeit und Lob. Die auditiven Hinweise der jeweiligen Buttons in den Bereichen können differenziert für einen bestimmten MWB oder für beide gleichzeitig vergeben und abgespielt werden (siehe Abbildung 4). Mögliche Hinweise sind das genauere Fahren oder Abkürzen als Kurvenverhalten, das Übernehmen oder Überlassen der Führung sowie dieselben Hinweise für Fahrtrichtung und Geschwindigkeit wie bei visuellen Hinweisen, allerdings auditiv. Als Letztes kann der Operateur an die MWB ein Lob vergeben, um z.B. die MWB über besonders gute Leistungen zu informieren. Nach der Vergabe von auditiven Hinweisen an die MWB sieht der Operateur, genau wie bei Vergabe von visuellen Hinweisen, ein Bild im Bereich der Streckenansicht am AMD. Die MWB werden allerdings nicht visuell informiert, damit sie auch einzeln instruiert werden können.

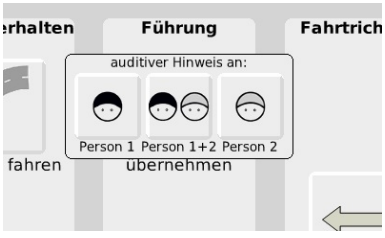


Abbildung 4: *AMD: Vergabe eines auditiven Hinweises an einen bestimmten MWB oder beide*

Unter harten Eingriffen versteht man die Eingriffe, die direkt in das Verhalten der beiden steuernden MWB eingreifen. So z.B. ist es dem Operateur möglich, den Grad der maximal-vertikalen Joystickumsetzung und damit die Gesamtgeschwindigkeit des Prozesses zu beeinflussen. Ein weiterer harter Eingriff ist die Möglichkeit, das Ausmaß der horizontalen Joystickumsetzung einzuschränken und damit das von den MWBn gesteuerte Objekt in eine bestimmte Richtung zu zwingen, z.B. an Gabelungen.

2.1.3 ATEO-Automation-Framework (AAF)

Das AAF ist eine Erweiterung zur Software-Komponente SAM und wurden im Zuge seiner Diplomarbeit von M. Hasselmann[5] entwickelt, um die automatisierte Regelung des Trackingprozesses mit Hilfe von Automatiken statt des Operateurs untersuchen zu können. Für die Zusammenstellung von Agenten[3] zu einer Automatik konnte ein Graphical User Interface (grafische Benutzeroberfläche) von E. Fuhrmann[3] entwickelt werden. Dieses GUI macht allen Benutzern den Automatik-Testmodus des ATEO-Projekts zugänglich, ohne dass tiefere Programmierkenntnisse vorausgesetzt werden müssen[4].
Es ist ebenso möglich, vorhandene Automatiken zu konfigurieren und durch ihre individuelle Zusammenstellung neue zu erschaffen. Die letzten Weiterentwicklungen des Automatiken-GUIs und AAF wurden von N. Kosjar[7] vorgenommen und bringen ein neues Ereignissystem und neue Automatiken mit sich. Der Vorteil des Ereignissystems ist, dass Agenten in Automatiken als externe XML-Dateien persistent abgespeichert werden. Die Automatiken sind dadurch portabel und ihr Graph in dem Automatiken-GUI vollständig rekonstruierbar. Außerdem können die Agenten-Aktivitäten durch die Einführung eines Ereignissystems auf bestimmte Bereiche vor und hinter Hindernissen, Kurven oder Gabelungen begrenzt werden.

[3] Diese Agenten führen die Automatik-Funktionen aus.

[4] Es ist durchaus möglich, eine Automatik direkt auf Programmcode-Ebene in Squeak/Smalltalk zu erstellen und zu konfigurieren, ohne dafür das Automatiken-GUI zu verwenden. Dieser Weg setzt jedoch entsprechende Kenntnisse des AAF und der Smalltalk-Programmierung voraus.

Das AAF macht also das Benutzen von Automatiken in der SAM möglich. Zum einen definiert das AAF Agenten und die Methoden, welche diese implementieren müssen, um als Bestandteil einer komplexeren Automatik zum Einsatz kommen zu können. Zum anderen stellt das AAF eine Datenstruktur zur Repräsentation von Graphen zur Verfügung, in welcher einzelne Agenten zu komplexen Automatiken verbunden werden können (siehe Abbildung 5).

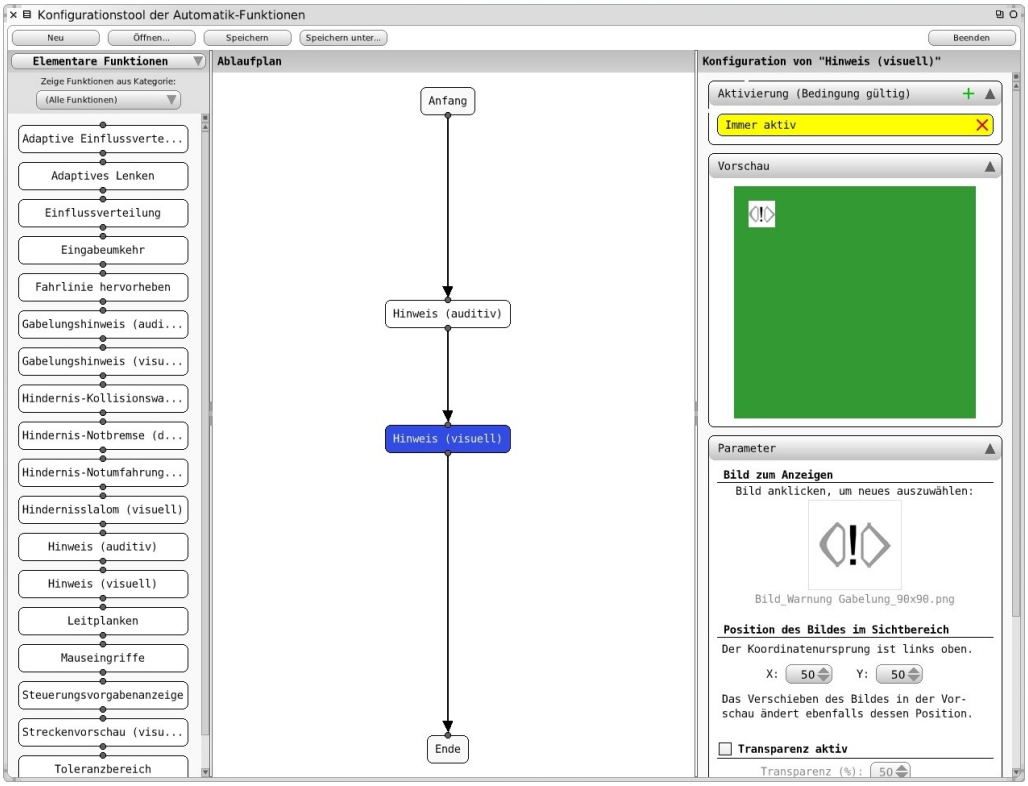


Abbildung 5: *AAF: Automatiken-GUI in aktueller Version mit zwei Agenten im Graphen; links: Agenten-Liste, mittig: Agenten-Graph, rechts: Agenten-Eigenschaften*

Dieser Verbund kann in das Automatiken-GUI eingelesen, verändert oder auch gespeichert (als *.aaf-Datei in XML-Notation) werden.
In der Textdatei „steps.txt“ der ATEO-Verzeichnisstruktur wird die Reihenfolge der Versuchsabschnitte festgelegt. Hier kann dann für jeden Abschnitt eine Automatik mit Hilfe des jeweiligen Automatik-Namen hinzugefügt werden.

Eine Bedienungsanleitung des Automatiken-GUI kann in der Studienarbeit von N. Kosjar[6], das Aktivieren von Automatiken im Versuch in der Diplomarbeit von E. Fuhrmann[3] oder beides im Anhang C nachgelesen werden.

2.1.4 ATEO-Lab-System (ALS)

Als ALS wird die Zusammenführung der Komponenten SAM, AAF, und AMD zu einer Versuchsumgebung bezeichnet. Es beinhaltet das AMD als Operateursarbeitsplatz (OA) und davon räumlich getrennt (schwarzer Balken in Abbildung 6) die SAM mit dem AAF als Mikrowelt, in der die MWB agieren. Als letzte Komponente sind 2 Computer zu erwähnen, an denen zum Ende eines Streckenabschnittes jeder der MWB seine Anstrengung einen bestimmten Skalenwert zuordnen soll. Der errechnete Wert wird dem Operateur ab Begin des folgenden Abschnittes im Statusanzeigenbereich des AMD (siehe Abbildung 2, links) angezeigt.

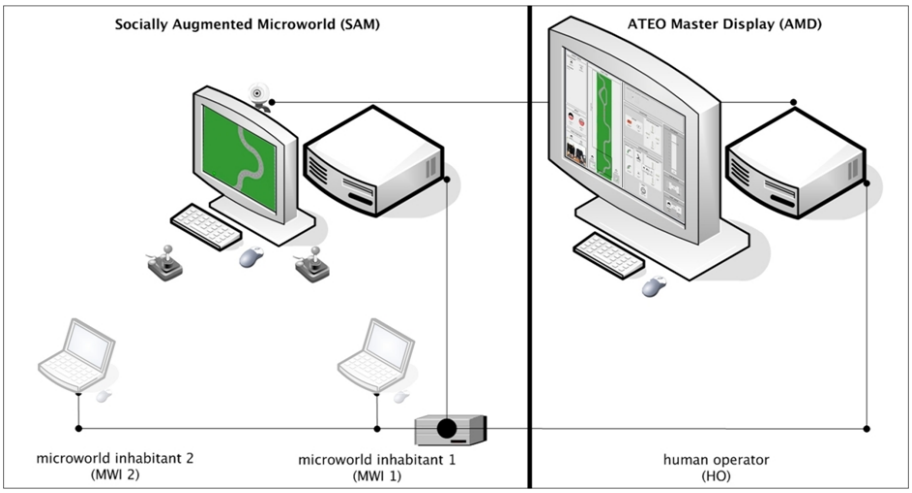


Abbildung 6: *ALS: SAM (links) mit Laptops - und davon räumlich getrennt - das AMD (rechts)*

Die Vernetzung von der SAM und dem OA ist inhaltlich aus zweierlei Hinsicht sinnvoll. Im Rahmen des ATEO Projektes soll ein leistungsbezogener Vergleich zwischen dem Operateur und den von Entwicklern entworfenen Automatiken gezogen werden.

Der Operateur hat die Aufgabe, diesen Trackingvorgang mittels AMD zu überwachen und optimierend einzugreifen. Dafür hat er neben den harten und weichen Eingriffen (siehe 2.1.2) zusätzliche Informationsquellen wie z.B. den Ausschlag der Joysticks oder die Streckenvorschau vorliegen sowie ein Live-Video der MWB (siehe Abbildung 2), an dem er die Gesichts- und Körpergesten nachvollziehen kann. Anhand der Auslastung der MWB könnte der Operateur nun z.B. entscheiden, ob er den Joystickinput ändert (von 50%-50% auf z.B. 30%-70%).

2.1.5 Logfile-Analysetool (LFA)

Das LFA ist ein Java-Programm, mit dem die anfallenden großen Testdaten untersucht und Informationen für die Auswertung vorbereitet und gefiltert werden. Die Auswertungsdaten befinden sich nach dem Versuchsdurchlauf in einem Unterordner, in welchem für jeden Versuchsabschnitt eine CSV-Datei und eine XML-Datei abgelegt werden. Die XML-Datei enthält Daten, wie dem Testmodus, der Versuchspersonennummer oder dem Geschlecht der MWB, die sich innerhalb des Versuches nicht ändern. In die CSV-Datei wird zu jedem Schritt (alle 39-40ms) des Versuchs ein Datensatz abgelegt, welcher unter anderem die Position des Tracking-Objekts, den Ausschlag der Joysticks und alle Operateurs-Aktionen festhält. Mit Hilfe des LFA-GUIs werden die CSV-Dateien eingelesen (siehe Abbildung 7) und mit Starten des Programms bearbeitet. Das Ergebnis der Logfile-Auswertung ist eine Microsoft Excel-Datei (mit der Dateiendung *.xls), in der die wichtigsten Informationen über den Trackingprozess und die Aktivitäten der MWB, der Agenten und des Operateurs enthalten sind. Um die Übersicht zu wahren, wurde diese Datei in mehrere themenspezifische Kategorien unterteilt. Als weitere Ausgabe sind Charts[5], also grafische Darstellungen von harten Eingriffen im Trackingprozess, zu nennen. Die aktuelle Version 1.9 wurde von A. Seid im Zuge seiner Studienarbeit[10] weiterentwickelt, in der nähere Informationen gefunden werden können.

[5]Diese Charts werden erst ab der LFA-Version 1.9 erstellt

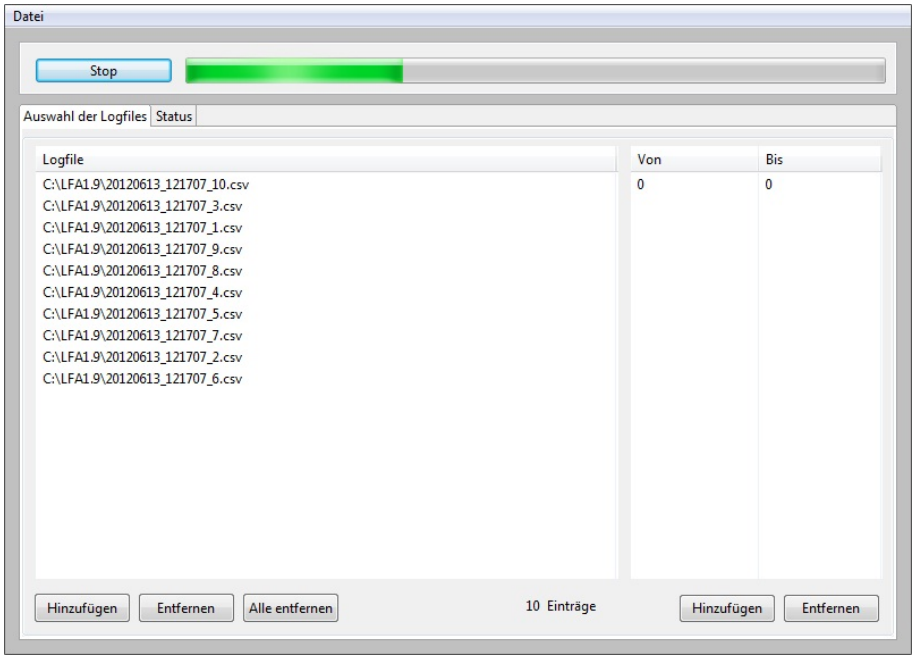


Abbildung 7: *LFA: Abarbeitung eines Satzes von Auswertungsdaten*

2.2 Extensible Markup Language (XML)

Die XML ist eine vom World Wide Web Consortium (W3C) spezifizierte Sprache zur strukturierten Darstellung und Beschreibung von Daten. Der Aufbau von XML-Dateien wird durch eine Schemasprache (z.B. DTD) individuell definiert und eingeschränkt. XML ist also anwenderspezifisch und wohlgeformt und wird im ATEO-Projekt beim Abspeichern und Laden des Automatiken-GUIs und beim Abspeichern der Logfiles verwendet. Die Dateien des Automatiken-GUIs enthalten unter anderem Informationen über die Konnektoren innerhalb des Graphen von der Quelle (root) bis hin zur Senke (sink), aber auch über eventuelle Verbindungen zu dazwischen liegenden Agenten als Knoten (node) (siehe Listing 1). Trotz ihres XML-Datenformates tragen sie die Dateiendung „*.aaf“. Außerdem wird für jeden Agenten eine Liste mit (zum Teil verschachtelten) auslösenden Ereignissen und seinen Eigenschaften sowie seiner Position und seinem Namen innerhalb des Graphen gespeichert.

```
<?xml  version="1.0" encoding="utf-8" ?>
<!DOCTYPE graph SYSTEM "aafgt.dtd">
<graph>
  <properties>
  </properties>
  <root>
    <id value="1"/>
    <label name="Anfang"/>
    <position x="289" y="17"/>
  </root>
  <node>
    <id value="3"/>
    <label name="Eingabeumkehr"/>
    <position x="0.5" y="0.47"/>
    <agent type="AAFInversionAgent">
      <activation>
        <event type="AAFAlwaysTrueEvent">
        </event>
      </activation>
      <properties>
        ... einige Konfigurationen
      </properties>
    </agent>
  </node>
  ... noch mehr Knoten
  <node>
  <sink>
    <id value="2"/>
    <label name="Ende"/>
    <position x="289" y="805"/>
  </sink>
  <connection parent="1" child="3"/>
  <connection parent="3" child="2"/>
</graph>
```

Listing 1: *XML am Beispiel eines Automatik-Graphen*

Bei der LFA wird XML für die Darstellung der Informationen des Experimentablaufes verwendet. Neben Werten über den Versuch an sich (z.B. Experimentnummer, Teamnummer) sind hier auch Informationen über die Anzahl der Streckenabschnitte und deren Streckenname abgebildet (siehe Listing 2).

```
<experiment>
  <experimentNr>1</experimentNr>
  <teamNr>1</teamNr>
  <mwi1>1</mwi1>
  <mwi2>1</mwi2>
  <assistanceNr>1</assistanceNr>
  <kindOfAssistance>OperatorAgents</kindOfAssistance>
  <gender>male</gender>
  <agentLogCodes>
    <directSetLogs>
      einige directSetLog-Codes
    </directSetLogs>
    <visualHintLogs>
      einige visualHintLog-Codes
    </visualHintLogs>
    <auditiveHintLogs>
      einige auditiveHintLog-Codes
    </auditiveHintLogs>
  </agentLogCodes>
  <stepInfo>
    <step Nr = "1">
      <mwMwiOneOnOP>0</mwMwiOneOnOP>
      <mwMwiTwoOnOP>0</mwMwiTwoOnOP>
      <trackname>testabschnitt</trackname>
      <inputMwi1>1.0</inputMwi1>
    </step>
    <step Nr = "2">
      <mwMwiOneOnOP>0</mwMwiOneOnOP>
      <mwMwiTwoOnOP>0</mwMwiTwoOnOP>
      <trackname>hauptabschnitt_lang</trackname>
      <inputMwi1>0.5</inputMwi1>
    </step>
  </stepInfo>
</experiment>
```

Listing 2: *XML-Notation am Beispiel eines ATEO-Logfiles*

2.3 Smalltalk und Squeak

Die einzelnen Softwarekomponenten, die im Versuchsablauf des ATEO-Projekts eingesetzt werden, sind in der Programmiersprache Smalltalk und seiner Open-Source-Implementierung Squeak (`http://www.squeak.org`) entwickelt.

In Smalltalk ist alles ein Objekt. Auch die Kontrollstrukturen wie bedingte Anweisungen oder Schleifen sind keine Konstrukte der Sprache selbst, sondern Eigenschaften bestimmter Objekte. Diese Konsequenz macht Smalltalk zu einer kleinen, eleganten, aber auch etwas esoterischen Sprache. Im eigentlichen gibt es nur drei eingebaute Konstrukte: zum Einen das Senden einer Botschaft an ein Objekt, zum Anderes das Zuweisen eines Objektes an eine Variable und als Letztes die Rückgabe des Objekts nach Ausführen einer Methode. Entwickelt wurde Smalltalk ab dem Ende der sechziger Jahre; die erste allgemein freigegebene Version von Smalltalk-80 erschien 1983.

Eine sehr angenehme und damals innovative Eigenschaft von Smalltalk ist die *dynamische Typisierung*. Demnach bestimmt der aktuelle Inhalt einer Variablen den Typ. Typangaben entfallen hier also, können aber andererseits nicht vom Compiler erkannt werden und zu Fehlern führen.[2]
Vorteilhaft für das schnelle Arbeiten mit Smalltalk ist die Eigenschaft, dass alle Änderungen ohne compilieren sofort aktiv sind.

Für den Aufbau des SAM- und des AMD-GUIs werden *Morphe* benutzt, die von Smalltalk bereitgestellt werden. Sie sind Objekte mit vielseitigen Eigenschaften (z.B. Farbe, Form, Position oder Titel) und können zum einen als Fenster für die Gruppierung anderer GUI-Objekte und Morphe oder zum anderen als Buttons (Schaltflächen) programmiert werden.

Diesen Button-Morphen müssen bei der Implementierung *ActionSelector*-Methoden zugewiesen werden. Diese sind Instanzmethoden einer Klasse, die ausgeführt werden, wenn die Schaltfläche betätigt wird.

2.4 Java

Java ist eine objektorientierte Programmiersprache, deren Entwicklung 1991 im Rahmen des Projekts Green bei dem Unternehmen Sun Microsystems begann. Java sollte einen einfachen Ersatz für C++ bieten, dabei jedoch die bekannte Syntax beibehalten. Im Januar 1996 wurde die erste Version (1.0) der neuen Sprache freigegeben und ist heute (momentan aktuelle Version: 7.0.5) ein Industriestandard.

Java verzichtet auf den radikalen, objektorientierten Ansatz von Smalltalk und stellt auch primitive Datentypen, die keine Objekte sind, zur Verfügung. Außerdem beherrscht Java das *Multithreading*, also die Möglichkeit, Programmteile quasi simultan nebeneinander zu betreiben.[2]

Verwendet wird Java ausschließlich bei der Analyse der anfallenden Logfiles. In einem Nebenprojekt von ATEO wurde allerdings begonnen, die SAM in Java zu überführen. Durch die Portabilität, Parallelisierbarkeit und Leistungsfähigkeit ist Java ideal geeignet, Programme für die Auswertung sehr großer Logfiles zu entwickeln. Für diesen Zweck bietet Java hervorragende vorgefertigte Bibliotheken zum Erstellen plattformunabhängiger GUIs und Grafiken in verschiedensten Datenformaten.

3 Das ATEO-Projekt und Implementierungen mit Frameworks

Dieses Kapitel stellt den theoretischen Teil der Diplomarbeit dar. Es werden zuerst einige ATEO-Komponenten als einzelne Frameworks definiert und vorher eine Einführung in das Thema gegeben. Im Anschluss werden die Programmierarbeiten als Framework-Implementierungen beschrieben.

Für die Implementierung der Anforderungen an diese Diplomarbeit und der damit verbundenen Änderungen an ATEO-Komponenten, waren Vorbereitungen nötig. Dafür mussten im Vorfeld die Programmiersprachen, die zum Einsatz kommen sollten festgelegt und eine davon unabhängige gute Architektur gefunden werden. Bei der Entwicklung der dritten Assistenzart als neuen Testmodus werden die einzelnen Komponenten des ATEO-Systems als Frameworks verwendet und vorher als solche klassifiziert.

3.1 Die Wahl der Programmiersprachen

Die Wahl der Programmiersprachen gehört zu den wesentlichen Bestandteilen der Vorbereitung von Implementierungen.

Im ATEO-Projekt sind die Programmiersprachen der einzelnen Komponenten bereits feststehend. Die Komponenten SAM zusammen mit dem AAF und das AMD als Benutzerschnittstelle des Operateurs sind in Smalltalk entwickelt und das Werkzeug LFA zur Auswertung der Versuchsdaten in Java. Alle Bestandteile können im Zuge dieser Diplomarbeit erweitert oder angepasst werden, so dass ein Wechsel der Programmiersprachen nicht zur Diskussion steht. Zur Zeit wird eine SAM-Version in Java überführt, wobei der Umfang mehrere Diplomarbeiten beträgt. Eine Neuimplementierung der restlichen Bestandteile in Java würde folglich den Umfang dieser Arbeit überschreiten.

3.2 Definition und Anwendung von Frameworks im Sinne des ATEO-Projekts

Um einen guten Softwareentwurf zu gewährleisten, wird oft auf die Verwendung von Rahmenwerken (Frameworks) anstelle von Bibliotheken zurückgegriffen. Bibliotheken bieten die Wiederverwendung von Funktionen an, die im Laufe eines Programms mehrmals gebraucht werden. Frameworks hingegen sind größere strukturierte Software-Pakete, die eine definierte Funktionsmenge abdecken. Sie müssen ausreichend abstrakt sein, um für möglichst viele Probleme eines Anwendungsbereiches eingesetzt werden zu können (Grechenig et al.[4]). Weiterhin wird eine Definition durch drei Punkte gegeben:

1. Ein Framework ist ein Applikationsskelett, welches vom Entwickler angepasst (spezialisiert) werden kann, um Anforderungen optimal umzusetzen.

2. Ein Framework ist ein wiederverwendbarer Entwurf. Dieser ist oft durch eine Menge von abstrakten Klassen und das Zusammenspiel ihrer Instanzen beschrieben.

3. Die Verwendung von Frameworks ist eine effiziente Art der Softwarewiederverwendung (Software-Reuse).

Ein Framework ist also ein erweiterbares System, bestehend aus zusammenarbeitenden Klassen und eine effiziente Art der Softwarewiederverwendung.

Nach einer weiteren gängigen Definition von H. Balzert[1] ist ein Framework ein "durch den Software-Entwickler anpassbares oder erweiterbares System kooperierender Klassen, die einen wiederverwendbaren Entwurf für einen bestimmten Anwendungsbereich implementieren". Sie sind spezifisch auf einen Anwendungsbereich ausgelegt und der Anwender des Rahmenwerks soll bei der Verwendung und Anpassung Unterklassen und Schnittstellen definieren.

Das ATEO-Projekt beinhaltet die Komponenten SAM, AAF, AMD und LFA, welche nun gegebenenfalls als Frameworks klassifiziert werden können, wenn sie die genannten Anforderungen und Definitionen erfüllen.

Vorab ist zu sagen, dass Smalltalk keine abstrakten Klassen (wie z.B. in Java) implementiert. Sie werden entweder durch den Klassennamen als solche definiert und hervorgehoben oder durch den Befehl `subclassresponsibility` innerhalb ihrer Methoden. Wird dieser Befehl in der Instanz- oder Klassenmethode einer „abstrakten“ Klasse verwendet, so müssen alle ableitenden Klassen diese Instanzmethode implementieren. Damit kann die Qualität der Software durch gute Strukturierung gesteigert werden.

SAM als Framework:
Die ATEO-Softwarekomponente SAM hat die Funktion den Ablauf des Versuchs zu steuern und ist als zusammenhängendes Rahmenwerk zu sehen. Sie enthält zwar keine abtrakten Klassen, dafür aber viele Instanzmethoden, die im Verlauf des Versuchs sukzessive abgearbeitet werden. Und doch ist die SAM durch viele Schnittstellen individuell benutzbar, wie das Ersetzen der Strecke oder der Form des Tracking-Objekts durch das Austauschen der entsprechenden Bilddateien. Für die Interaktion mit anderen Frameworks des ATEO-Projekts werden ausschließlich Schnittstellen verwendet, um eine klare Abgrenzung zu wahren.

AAF als Framework:
Das AAF ist mit seinem Automatiken-GUI für die Zusammenstellung von Agenten zu einer Automatik zuständig. Die Agenten sind hierbei Bestandteil des Automatik-Graphen und müssen bestimmte Eigenschaften erfüllen. So benötigen sie die Instanzmethode `compute`, an welche bei der Abarbeitung des Graphen ein *SamState*, also ein Abbild der Zustandsvariablen von der SAM, die in der Klasse SAMModelData hinterlegt sind, übergeben wird. Im Anschluss kann dieses Abbild verändert und zurückgegeben werden. Jeder Agent leitet deshalb von der abstrakten Klasse AAFAgent ab, die das Integrieren der `compute`-Methode mittels `subclassresponsibility` vorschreibt.

Diese abstrakte Klasse gibt weiterhin die Einbindung der Klassenmethode `plaintextName` vor, welche der Agenten-Klasse einen Namen im Automatiken-GUI zuweist.
Die Komponenten SAM und AAF sind eigenständige Rahmenwerke und kommunizieren ausschließlich über die Schnittstelle *SamState* miteinander. Aber auch das AAF greift auf externe Daten zurück. So werden die Eigenschaften von Automatiken nicht im Programmcode festgehalten, sondern über spezielle XML-Dateien eingelesen, welche vorher mit dem Automatiken-GUI zusammengestellt und abgespeichert wurden.

AMD als Framework:
Die Funktion des AMDs innerhalb des ATEO-Systems ist es, dem Operateur eine Benutzeroberfläche bereitzustellen, mit welcher er die Vorgänge in der SAM überwachen und bei Bedarf eingreifen kann. Das AMD kommuniziert als eigenständiges Rahmenwerk im Operateur- und im neuen „Opermatik"-Modus ausschließlich über die Netzwerk-Schnittstelle mit der SAM, wodurch es ebenso als Framework zu klassifizieren ist. Es enthält einige abstrakte Klassen, von denen z.B. GUI-Elemente ableiten. Dies können neben Buttons, bei deren Implementierung bestimmte Richtlinien wie der Übergabe der Button-Größe, seiner Position oder einer *ActionSelector*-Klasse eingehalten werden müssen, auch andere Morphe sein, die zum Aufbau des AMDs zum Einsatz kommen.

LFA als Framework:
Das Werkzeug LFA ist für die Aufbereitung der Log-Daten zuständig und für den eigentlichen Versuchsablauf nicht notwendig. Es ist ein eigenständiges, abgeschlossenes System und die einzige Versuchskomponente, die in Java programmiert ist. Es ist bereits spezialisiert und dadurch die weitere Nutzung in anderen Systemen nicht möglich. Da weiterhin die Auswertung der Testdaten im Java-Programm über mehrere Bibliotheken und nicht über eigene Frameworks abgewickelt wird, ist das LFA nicht als Framework, sondern als eigenständige Software-Komponente des ATEO-Systems zu sehen.

3.3 Der Einsatz von Frameworks bei der Implementierung der Anforderungen

In größeren Projekten werden Frameworks verwendet, um Kosten durch eine Wiederverwendung von Software-Komponenten zu senken und dabei die Qualität des zu entwickelnden Produktes hoch zu halten. Dabei werden die Kosten nicht zuletzt niedrig gehalten, weil die Umsetzung der Anforderungen an das Projekt durch Framework-Implementierungen auch zeitlich geringer gehalten werden kann als bei Neuimplementierungen.

Auswahl der Frameworks:
Bevor Frameworks bei der Umsetzung der Anforderungen eingesetzt werden können, ist es unumgänglich, die notwendigen sorgfältig aus einer Menge von vorhandenen Frameworks auszuwählen und zu begutachten, denn das spätere Austauschen der Frameworks ist schlechtestenfalls nur mit sehr großem Aufwand realisierbar. Die Auswahl ist für die Umsetzung der Anforderungen an den Versuchsablauf in dieser Diplomarbeit trivial, da nur drei Frameworks zur Verfügung stehen und deren Funktionen integriert und teilweise editiert werden können.

In anderen Software-Projekten stehen üblicherweise eine Vielzahl von Frameworks zur Verfügung, die neben der Abdeckung von Anforderungen auch auf Wartbarkeit, Erweiterbarkeit und auf die uneingeschränkte Nutzung in Bezug auf die Kommerzialisierung des Frameworks untersucht werden müssen (Grechenig et al.[4]). Für den ATEO-Versuchsablauf decken die drei Frameworks SAM, AAF und AMD nach deren Erweiterung die Anforderungen ab und sind uneingeschränkt nutzbar. Die Wartbarkeit ist durch die Editierung einer Vielzahl von Entwicklern nur beschränkt gegeben und der Programmcode durch verschiedene Programmierstile und -aufgaben teilweise undurchsichtig. Dennoch konnten alle Frameworks nach einigem Zeitaufwand angepasst und verbunden werden.

Framework-Implementierungen des ATEO-Systems:
Die Nutzung des neuen „Opermatik“-Modus erfordert das Erweitern der Netzwerk-Schnittstelle in SAM, um dann über eine weitere Schnittstelle (den *SamState*) mit den Agenten, die das AAF verwaltet, kommunizieren zu können. Diese Netzwerk-Schnittstelle wird in eigenen Instanzmethoden und einer eigenen Klasse umgesetzt, so dass die Frameworks SAM und AAF in ihrer Struktur unverändert bleiben. Ein Abgrenzungskriterium lässt hierbei auch eine eventuelle Erweiterung des AAF-Frameworks zu. Dies war mit der Umsetzung der Anforderungen sogar notwendig, und das AAF musste um zwei Ereignisse erweitert werden. Die Klassenstruktur bleibt hier aber unverändert, da die beiden Ereignisse von der abstrakten Klasse AAFAbstractEvent ableiten.

Durch die Entwicklung der neuen Schaltflächen am AMD, die nicht im Operateur-Modus verwendet werden können, musste das Framework für den „Opermatik“-Modus spezialisiert werden. So entstand als neues Framework die Version 3.0 des Operateursarbeitsplatzes, welcher spezielle Buttons der abstrakten Klasse OPButtonWithMouseAction integriert hat, die unter anderem die Übergabe zweier *ActionSelector*-Methoden für die verschiedenen Ausführungen bei der Benutzung von zwei Maustasten fordert.

Anpassungen an das LFA:
Alle Erweiterungen des LFAs werden in neu entwickelten Klassen vollzogen. Da diese die Auswertung von Datensätzen aller Versuchsmodi sicherstellen soll, können neben dem Entwurf auch alle Code-Segmente wiederverwendet werden und nicht nur einige Unterklassen. Das LFA ist nach Abschluss der Arbeiten und den damit verbundenen Erweiterungen von der Version 1.9 in die Version 2.0 überführt worden.

4 Die Erweiterung des AMDs um visuelle und auditive Agenten

In diesem Kapitel werden zuerst die Anforderungen an diese Arbeit detailliert und verbal beschrieben. Dem folgen die Anpassungen an das SAM-, AAF- und das AMD-Framework. Als Letztes werden die Resultate der Auswertung von Versuchsdaten durch das LFA und die Tests der entwickelten Funktionen vorgestellt.

4.1 Anforderungen

„Zu den wichtigsten Tätigkeiten innerhalb des Software-Entwicklungsprozesses gehören das Definieren der Produkt-Anforderungen und die Modellierung der fachlichen Lösung“(H. Balzert)[1].
Die Anforderungen an das zu entwickelnde Produkt werden hier definiert und in drei Kategorien unterteilt. Unter Musskriterien werden die Leistungen an das Produkt gezählt, die dringend notwendig sind, um die grundlegenden Anforderungen an das Produkt zu erfüllen. Als Wunschkriterien werden Leistungen bezeichnet, die nicht unabdingbar sind und vom Entwickler so gut wie möglich umgesetzt werden sollten. Abgrenzungskriterien hingegen sind Ziele, die das entgültige Produkt nicht leisten können soll.

4.1.1 Musskriterien

Erstellung und Integration eines neuen Modus:
Die Erweiterung des ATEO-Projekts in dieser Diplomarbeit bringt einen neuen Testmodus mit sich, der sich als Zusammenschluss der zwei bisherigen Testarten sehen lässt. Da der Testmodus für Versuche am SAM-GUI eingestellt werden kann, muss dieses um den neuen Modus ergänzt werden. Es ist ein geeigneter, aussagekräftiger Name für den Modus zu finden.

Starten des neuen Modus:
Für den neuen Modus ist es notwendig, dass eine Netzwerkverbindung zwischen den Rechnern des Operateurs und der MWB besteht. Ist dies nicht der Fall, so muss eine Zustandsmeldung erfolgen, welche auf die Art des Problems aufmerksam macht.

Zusammenstellen von Netzwerk-Agenten:
Die Agenten, die dem Operateur zur Verfügung gestellt werden sollen, müssen im Agenten-GUI von bisherigen Agenten zu unterscheiden sein, sodass sie für spezielle Versuche leicht konfiguriert werden können. Demnach müsste das GUI also so überarbeitet werden, dass die Klassifizierung eines beliebigen Agenten als Netzwerk-Agent über eine einfache Schaltfläche möglich ist und der Benutzer somit nur einen geringen Aufwand hat.

Schnittstellen AMD $\leftrightarrow$ SAM:
Um Agenten vom OA ansprechen zu können, muss die Kommunikation der Frameworks über das Netzwerk erweitert werden. Hierfür ist es notwendig, eine geeignete, kompakte Darstellung der Informationen zu finden, die den Netzwerkverkehr so gering wie möglich zusätzlich belastet. Die Informationen der Schnittstelle AMD $\rightarrow$ SAM stellen dabei die Button-Aktivitäten des AMD-GUIs durch den Operateur dar. Die Schnittstelle SAM $\rightarrow$ AMD beinhaltet die Aktivitäten der dazugehörigen Netzwerk-Agenten.

Aktivierung der Agenten am AMD:
Die Aktivierung der Agenten am AMD soll über dessen Schaltflächen ermöglicht werden. Wird ein Button aktiviert, so soll ein gelber Rahmen um ihn gelegt werden, mit dem die Aktivierung gut sichtbar wird. Führt ein Agent eine Aktion[6] aus, so muss der Operateur ebenso darüber informiert werden. Aktive Agenten-Buttons oder Agenten-Hinweise sollen vor Beginn einer jeden Teststrecke zurückgesetzt werden, sodass der gleiche Ausgangsstatus für jeden Testabschnitt gegeben ist. Weiterhin ist die Benutzung von Automatiken nicht an Streckenabschnitte gebunden. Abschnitte ohne Automatiken werden demzufolge mit einer Dummy[7]-Automatik belegt.

Datenerhebung:
Die Informationen über die Aktivitäten des Operateurs und damit auch der Netzwerk-Agenten müssen verlustlos aufgezeichnet werden, um sie später fehlerfrei auszuwerten. Dafür müssen die bisherigen Auswertungsdaten im Logfile effizient erweitert werden, damit die Logfile-Dateien nicht zu groß werden, aber auch der Verlust von Daten vermieden wird.

Logfileanalyse:
Das LFA ist an die veränderten Datensätze so anzupassen, dass auch ältere Auswertungsdatensätze ohne die Informationen über die Netzwerk-Agenten-Aktivitäten analysiert werden können. Die neuen angepassten Datensätze sollten bei der Analyse in die bisherige resultierende Datei integriert und in eindeutiger Darstellung gespeichert werden.

Squeak und Smalltalk:
Da alle relevanten Bestandteile des ALS in Smalltalk und seiner speziellen Entwicklungsumgebung Squeak entwickelt wurden, soll dies für die Fertigstellung des zu entwickelnden Produkts beibehalten werden.

[6] z.B. die visuelle oder auditive Hinweisvergabe an die MWB
[7] Diese Automatik enthält keine Agenten. Die Quelle ist also direkt mit der Senke verbunden und die Daten werden ohne Manipulierungen durchgereicht.

Erweiterbarkeit des Systems:
Es ist bis jetzt noch nicht absehbar, in welche Dimensionen sich das ATEO-System entwickelt. Mit der Entwicklung eines dritten Modus sollen Agenten vom Operateur angesteuert werden können. Diese Erweiterung wird die Integration einiger Agenten betreffen, soll aber andererseits nicht die Weiterentwicklung und damit die Integration anderer Agenten einschränken. Die neuen Netzwerk-Informationen sollten leicht erweiterbar sein und einen hohen Informationsgehalt annehmen können.

4.1.2 Wunschkriterien

Nicht-Netzwerk-Agenten:
Im neuen dritten Modus ist es vorgesehen, Netzwerk-Agenten vom Operateur hinzuschalten zu können. Das zu entwickelnde Produkt sollte hier unterscheiden, ob auch Agenten während des gesamten Durchlaufes aktiviert sein können. Dies sind also Agenten, die nicht vom Operateur ansteuerbar sind[8].

Das Ereignissystem und Agenten:
Da das vorhandene Ereignissystem genutzt werden soll, ist es wünschenswert, die verschiedenen Ereignisse zu überprüfen und im speziellen Fall anzupassen. Die ebenso verwendeten Agenten sind im Sinne der Entwicklungen auf ihre Eigenschaften zu untersuchen.

Charts vom LFA:
Neben der Auswertung der Logfiles als tabellarische Darstellung kann auch eine visuelle Ausgabe der Button- und Agenten-Aktivitäten als Ergebnis entstehen. Es sollte eine Darstellung gewählt werden, in der die Aktivitäten aller Agenten einer Automatik für jede Teststrecke auf einen Blick sichtbar sind. Außerdem sollten Gabelungen gut sichtbar in die Ausgabe integriert werden.

[8] Ein Beispiel hierfür wäre ein Agent, der eine Streckenansicht für die MWB in der SAM einblendet, die so immer aktiv wäre.

4.1.3 Abgrenzungskriterien

Benutzung von Netzwerk-Agenten:
Für die Zusammenstellung von Netzwerk-Agenten zu einer Automatik und deren anschließenden Einsatz sind bereits ausgereifte und benutzbare Agenten des Ereignissystems zu verwenden. Im Zuge der Erfüllung und Umsetzung der Aufgaben dieser Diplomarbeit ist eine Erweiterung des Ereignissystems möglich.

Aktivierung der Agenten am AMD:
Die Aktivierung der Agenten am AMD soll ausschließlich über die bereits bestehenden Schaltflächen ermöglicht werden, um die Vergleichbarkeit zu vorherigen Operateur-Tests ohne Netzwerk-Agenten zu wahren. Es dürfen also keine optischen Veränderungen (wie z.B. das Verschieben von Schaltflächen oder zusammenhängenden Bereichen) vorgenommen werden, sondern nur funktionelle Anpassungen der Buttons selbst. Die Agenten sollen nur visuelle oder auditive Funktionen des Operateurs automatisieren.

Dominanz von visuellen Hinweisen:
Im von N. Kosjar[7] entwickelten Ereignissystem ist keine Dominanzfolge von visuellen Anzeigen implementiert[9]. Die Entwicklung eines solchen Dominanzverhaltens würde eine grundlegende Überarbeitung der Abarbeitung des AAF-Graphen nach sich ziehen und den Umfang dieser Diplomarbeit überschreiten.

Aktivierung der Agenten allgemein:
Das Aktivierungsintervall der Agenten weist im Gegensatz zum Operateur ein Echtzeitverhalten und kein festgelegtes Zeitintervall auf. Die Umstellung der zeitlichen Abarbeitung der Agenten-Hinweise im vorhandenen Ereignissystem würde den Umfang dieser Diplomarbeit überschreiten.

[9] wie z.B. die Dominanz der Anzeige von visuellen Hindernishinweisen gegenüber Gabelungshinweisen

4.2 SAM+AAF

In diesem Unterkapitel werden die Erweiterungen an das SAM- und das AAF-Framework beschrieben. Besonders umfassend sind dabei die Ergänzungen der Schnittstellen, des Automatiken-GUIs und des Ereignissystems.

4.2.1 Implementierung der Schnittstellen

Definition Schnittstelle:
Nach H. Balzert[1] wird in der objektorientierten Softwareentwicklung der Begriff Schnittstelle (interface) nicht einheitlich verwendet. „In der Regel definiert eine Schnittstelle Dienstleistungen für Anwender, d.h. für aufrufende Klassen, ohne etwas über die Implementierung der Dienstleistungen festzulegen."
Die Aufgabe der Schnittstelle in SAM ist es, den Teil der Netzwerkpakete, der die Informationen über die Button-Zustände des AMD als Zeichenkette (String) enthält, zu isolieren und ihn für verschiedene Klassen zur Verfügung zu stellen. Zusätzlich ist die Schnittstelle dafür zuständig, dass die Agenten-Aktivitäten des AAF in einem String zusammengefasst und codiert, sie an den ausgehenden Netzwerk-Verkehr angeheftet und abgeschickt werden.
Die Strings haben also zwei verschiedene Aufgaben, sind in ihrer Grundform allerdings gleich. Es sind jeweils dreißigstellige Strings, wobei jede der Stellen eine durchnummerierte Position darstellt. Der Wert jeder Stelle beinhaltet Informationen über entweder einen Buttonzustand des AMD oder aller Agenten, die diese als Netzwerk-ID (Nummer) zugewiesen bekommen haben. Im Zuge dieser Diplomarbeit wurden die Werte der einzelnen Stellen nur mit den Zahlen Null (für inaktiv) oder Eins (für aktiv) codiert. Man kann hier also von einer dreißigstelligen Binärzahl sprechen. Außerdem sind nur zehn der dreißig Stellen besetzt (siehe Tabellen 1 - 3), da bis jetzt kein weiterer Bedarf an Unterstützungen des Operateurs durch Netzwerk-Agenten besteht. Andererseits ist es leicht möglich, die verbleibenden Stellen zu belegen und deren Werte höher zu codieren. So kann das ATEO-System im neuen Testmodus nahezu uneingeschränkt erweitert werden.

Pos.	Fkt. der Agenten	Wert
1	vis. Gabelung	0/1
2	vis. Hindernis	0/1
3	vis. schneller	0/1
4	vis. langsamer	0/1
5	vis. links	0/1
6	vis. rechts	0/1
7	-	0
8	-	0
9	-	0
10	-	0

Tabelle 1: *Belegungen 1 bis 10 (visuelle Automatiken)*

Pos.	Fkt. des Agenten	Wert
11	-	0
12	-	0
13	aud. schneller	0/1
14	aud. langsamer	0/1
15	aud. links	0/1
16	aud. rechts	0/1
17	-	0
18	-	0
19	-	0
20	-	0

Tabelle 2: *Belegungen 11 bis 20 (auditive Automatiken)*

Pos.	Fkt. des Agenten	Wert
21 .. 30	-	0

Tabelle 3: *Belegungen 21 bis 30 (andere Automatiken)*

Isolation der Informationen aus dem Netzwerkstring:
In der Instanzmethode `processNetwork` der Klasse SAMControllerNetwork werden die Informationen des eingehenden Netzwerk-Pakets tokenisiert, also mit Hilfe eines Trennzeichens geteilt, und an entsprechende Methoden übergeben. Der Token für den Netzwerk-String befindet sich hier an Stelle zwölf (siehe Listing 3) und wird an die Methode `processAutomationCode` weitergereicht.

```
1 processNetwork
2
3 ...
4   self processMentalWorkloadMwiOne: (tokens at: 10).
5   self processMentalWorkloadMwiTwo: (tokens at: 11).
6   self processAutomationCode: (tokens at: 12).
7 ...
```

Listing 3: *Die Instanzmethode processNetwork der Klasse SAMControllerNetwork*

Die Instanzmethode `processAutomationCode` nimmt die Informationen entgegen und reicht sie an die Methode `networkData` der Klasse SAMModelData weiter (siehe Listing 4). Diese dient zum einen dazu, alle wesentlichen Informationen des Trackingvorgangs zu speichern, um sie für andere Instanzmethoden zugänglich zu machen. Zum anderen wird diese Klasse als Schnittstelle vom SAM- und AAF-Framework verwendet. Zusätzlich ist zu sagen, dass die Methode `processAutomationCode` sehr effektiv arbeitet, da sie die übergebenen Informationen nur weiterreicht, wenn sich deren Zustand geändert hat.

```
1 processAutomationCode: networkString
2
3 (networkString = newstring)
4 ifFalse:[
5     modelData networkData: networkString.
6     newstring := networkString.
7 ].
```

Listing 4: *Die Instanzmethode processAutomationCode der Klasse SAMControllerNetwork*

Die Ansteuerung und Arbeitsweise der Agenten:
Im AAF wird ein Abbild von einzelnen Zuständen und Werten der aktuellen Trackinginformationen aus der Klasse SAMModelData angelegt, der *SamState*. Dies ist notwendig, damit die einzelnen Agenten dieses Abbild sukzessive manipulieren können. Für jede Agenten-Klasse muss dafür eine Instanzmethode angelegt sein (die sogenannte `compute`-Methode), die bei der Abarbeitung des Automatik-Graphen einen *SamState* übergeben bekommt und anschließend wieder zurückgibt. Die Verwaltung der Übergaben an die einzelnen Agenten als Knoten im Graphen übernimmt die `compute`-Methode der Klasse AAFNode. Sie unterscheidet zwischen Nicht-Netzwerk-Agenten und aktiven- sowie inaktiven Netzwerk-Agenten, wobei das Übergeben des *SamState* an inaktive Netzwerk-Agenten unterdrückt wird (siehe Listing 5).

```
compute: aSamState

| networkActive networkID networkString |

networkActive:= delegate networkIDActive.
networkID:= delegate networkID.
networkString:= aSamState trackingState networkData asString.

networkActive
ifFalse: [
    self result: (self delegate compute: aSamState)
  ]
ifTrue: [
    ((networkString ~= '0') and: [(networkString at: networkID) == $1])
    ifTrue:[
      self result: (self delegate compute: aSamState)
      ].
  ].
```

Listing 5: *Die Instanzmethode compute der Klasse AAFNode*

Die Unterdrückung führt dazu, dass diese Agenten keinen Einfluss am Abbild nehmen können, also nicht aktiv sind.

4.2.2 Entwurf der Anpassung des Automatiken-GUIs

Das von N. Kosjar[7] weiterentwickelte Automatiken-GUI bietet dem unerfahrenen Anwender die Möglichkeit, Agenten zu einer Automatik zusammenzustellen, ohne die XML-Datei manuell erstellen zu müssen. In diesem GUI können für jeden einzelnen Agenten individuelle Einstellungen vorgenommen werden (siehe Abbildung 5, rechts). Dazu gehören unter anderem ein aktivierendes Ereignis, eine Vorschau und diverse Parameter.

Hier wurde eine neue Möglichkeit zur Konfiguration von Agenten geschaffen: die Netzwerk-ID als Teil des Netzwerk-Strings. Sie ist zentraler Bestandteil der Kommunikation zwischen Agenten des AAFs und den Buttons des AMDs und eine feste Konstante in Abhängigkeit der Button-Nummern. Für das Einstellen und Benutzen dieser Nummer wurde dem Konfigurationsmorph ein Submorph hinzugefügt, mit dem sich diese Zahl editieren lässt.

Um einen Agenten als Netzwerk-Agenten zu deklarieren, muss in dessen Konfigurationsleiste die Checkbox[10] „Als Netzwerkagenten aktivieren“ aktiviert werden. Der darunterliegende Bereich für das genaue Einstellen der Netzwerk-ID ist dann nicht mehr ausgegraut (siehe Abbildung 8). Mittels der beiden Dreiecke neben dem Zahlenfeld kann jetzt eine Nummer zwischen eins und dreißig ausgewählt werden. Der resultierende Effekt ist, dass der Agent nur aktiv wird, wenn der entsprechende Button am AMD aktiv ist. Um eine gute Gebrauchstauglichkeit des GUIs zu erreichen, wurde ein Tooltipp[11] eingebunden, der den Wertebereich (1-30) anzeigt.

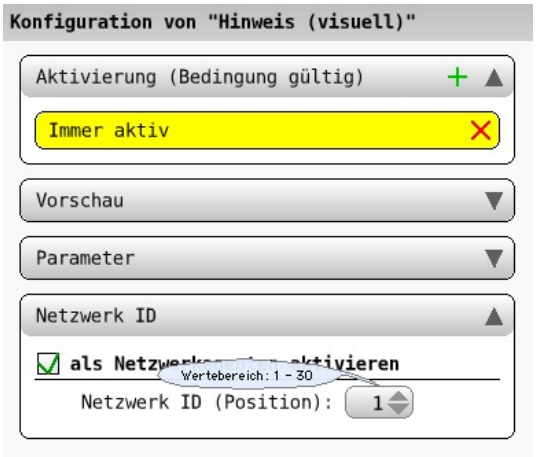


Abbildung 8: *Automatiken-GUI: neue Option „als Netzwerkagenten aktivieren“, Vergabe der ID und Anzeige eines Tooltipps.*

Da jeder Agent als Netzwerk-Agent konfiguriert werden können soll[12], erscheint dieser Konfigurationsmorph immer dann, wenn ein einzelner Agent auf niedrigster Ebene ausgewählt wird. Wird eine Automatik zusammengefasst im Graphen angewählt, so muss erst der Detailgrad erhöht werden, ähnlich wie bei den Parametern (siehe Abbildung 9).

[10] Eine Checkbox ist ein Standardbedienelement, mit dem Ja/Nein-Fragen beantwortet werden können.

[11] Tooltipps sind Popups, die einen kurzen erklärenden Text anzeigen, wenn der Benutzer mit dem Mauszeiger über einen Bereich des GUI fährt.

[12] und zwar auch partielle Agenten innerhalb einer Automatik

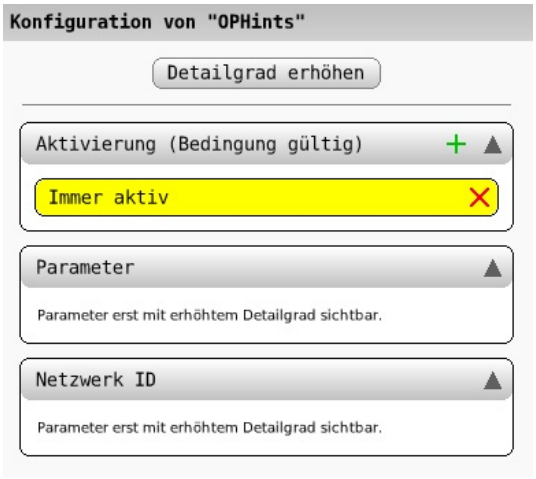


Abbildung 9: *Automatiken-GUI: Detailgrad zu niedrig und resultierender Hinweis: „Parameter erst mit erhöhtem Detailgrad sichtbar“.*

Netzwerk-ID und XML:
Die Einstellung der Netzwerk-Agenten im AAF-GUI ziehen neue Informationen der einzelnen Automatiken nach sich, die in der für die Speicherung der Automatiken vorgesehenen XML-Dateien festgehalten werden müssen. Der Zustand (engl. value= Wert) der Checkbox des Netzwerk-ID-Morphs findet sich im Eigenschaftenbereich `prop key=‘‘networkIDActive‘‘` der XML-Datei wieder; die anschließend eingestellte Zahl, also die Netzwerk-ID, im `prop key=‘‘networkID‘‘` (siehe Listing 6).

```
...
    <agent type="AAFInversionAgent">
      <activation>
        <event type="AAFAlwaysTrueEvent">
        </event>
      </activation>
      <properties>
        <prop key="networkIDActive" value="true"/>
        <prop key="networkID" value="3"/>
      </properties>
    </agent>
...
```

Listing 6: *Erweiterung der XML-Dateien*

Die Behandlung früherer Agenten:
Jeder neue Agent erhält automatisch in der XML-Datei seiner Automatik diese zwei Eigenschaften. Fehlen diese (z.B. in einer früheren XML-Datei), so bleiben sie standardmäßig auf `false` (NetworkIDActive) und `0` (NetworkID). Beinhaltet der Agent beim Laden allerdings diese Eigenschaften, dann werden diese Standardwerte überschrieben (siehe Listing 7).

```
1  setAllProps: aPropertyDictionary
2  ...
3    super setAllProps: d.
4
5    (d includesKey: 'networkID') & (d includesKey: 'networkIDActive')
6    ifTrue:[
7      self networkID:
8        (AAFUtils convert: (d at: 'networkID') type: 'Number').
9      self networkIDActive:
10       (AAFUtils convert: (d at: 'networkIDActive') type: 'Boolean').
11     ].
12 ...
```

Listing 7: *Methode setAllProps der Klasse AAFAgent*

Wird ein Agent ohne diese Eigenschaften in das Automatiken-GUI geladen, manipuliert und gespeichert, so wird dieser Agent automatisch aktualisiert und mit diesen zwei Eigenschaften versehen. Der überarbeitete Einlesevorgang ist also abwärtskompatibel zu früheren Agenten.

4.2.3 Die Automatik „OPAutomatik“

Die Automatik „OPAutomatik“ (OP = Operateur) ist eine Automatik, die speziell für die Umsetzung dieser Diplomarbeit angefertigt wurde und besteht im Wesentlichen aus sechzehn Agenten: zwölf visuellen und vier auditiven. Den visuellen Agenten sind sechs Buttons des AMD zugeteilt. Ein Button steuert also zwei Agenten gleichzeitig an. Grund dafür ist die Wahrung der Vergleichbarkeit zu manuellen Hinweisen des Operateurs. Gibt dieser nämlich einen Hinweis an die MWB, so erscheinen zwei Morphe auf deren Bildschirm (siehe Abbildung 3).

Im überarbeiteten Automatiken-GUI von N. Kosjar[7] ist es nur möglich, genau einen Hinweis von einem Agenten anzeigen zu lassen. Mit der Vergabe des gleichen Hinweises von zwei Agenten an derselben Stelle wie die manuellen Hinweise vom Operateur, kann dieses Verhalten ideal imitiert werden. Einer der Agenten zeigt den Hinweis bei den Koordinaten -50@-314 und der andere bei -660@-314 an, genau wie die Hinweise des Operateurs. Weiterhin sind die angezeigten Hinweisbilder der Agenten die gleichen, die auch bei der Vergabe von manuellen Hinweisen des Operateurs in der SAM angezeigt werden.

Die Agenten haben folgende Aktivierungsvoreinstellungen im Ereignissystem:

ID	Agentenname	Ereignis	von	bis	Anz. L/R
1	visFork	Gabelung	-800	-250	L
1	visFork2	Gabelung	-800	-250	R
2	visObstacle	Hindernis	-1000	-370	L
2	visObstacle2	Hindernis	-1000	-370	R
3	visForw	Strecke schnell fahren	-	-	L
3	visForw2	Strecke schnell fahren	-	-	R
4	visBack	Strecke langsam fahren	-	-	L
4	visBack2	Strecke langsam fahren	-	-	R
5	visLeft	Gabelung&ML	-250&L	50&L	L
5	visLeft2	Gabelung&ML	-250&L	50&L	R
6	visRight	Gabelung&ML	-250&R	50&R	L
6	visRight2	Gabelung&ML	-250&R	50&R	R
13	audForw	Strecke schnell fahren	-	-	-
14	audBack	Strecke langsam fahren	-	-	-
15	audLeft	Gabelung&ML	-250&L	50&L	-
16	audRight	Gabelung&ML	-250&R	50&R	-

Tabelle 4: *Agenten der Automatik „OPAutomatik“ (vis = visuell, aud = auditiv, Fork = Gabelung, Obstacle = Hindernis) mit auslösenden Ereignissen (ML = Mittellinie) und Hindernisanzeige (L = links; R = rechts)*

Diese Voreinstellungen sind ausschließlich empirische Werte. Sie resultieren aus der Abfolge der maximal möglichen, hintereinander folgenden Ereignisse. Mit diesen Werten wird vermieden, dass das gleichzeitige Anzeigen zweier

Hinweise eintritt. Das Verhalten in einem solchen Fall kann vom Ereignissystem nicht eindeutig und konsequent abgearbeitet werden. Ein Beispiel für die Umgehung des Problems wird in Abbildung 10 dargestellt.

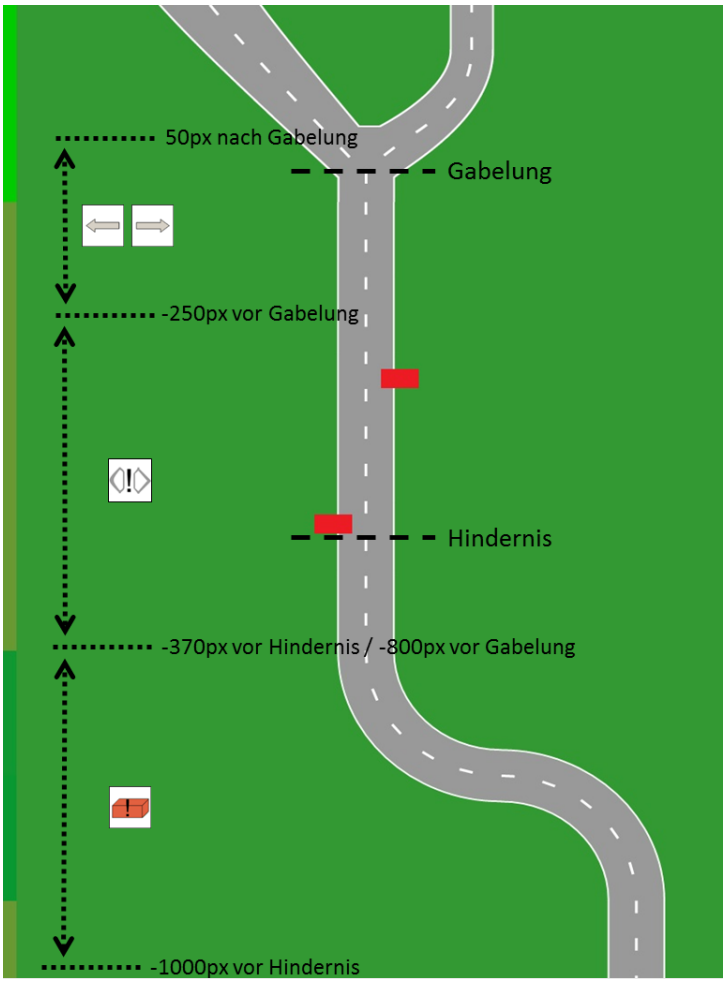


Abbildung 10: *Max. Sequenz aus Ereignissen (Hindernis-Gabelung-Kombination) mit Abfolge der Anzeige von Agenten-Hinweisen (links)*

In Abbildung 10 wird veranschaulicht, dass eine visuelle Hinderniswarnung 1000 bis 370 Pixel vor dem Hindernis präsentiert wird. Die obere Grenze (370 Pixel) ist die gleiche Grenze, wie die untere Grenze (800 Pixel) der Gabelung, an dem der Gabelungs- und Richtungshinweis festgesetzt ist. Somit überschneiden sich die Hinweisanzeigen nicht.

Das gleichzeitige Eingreifen mehrerer Agenten in der SAM: Durch das Setzen der Agenten-Einstellungen wird vermieden, dass zwei visuelle Anzeigen gleichzeitig in der SAM aufgezeigt werden können. Wenn allerdings ein visueller und ein auditiver Agent gleichzeitig eingreifen, so wird der Hinweis des visuellen Agenten angezeigt und der Ton des auditiven Agenten zur selben Zeit abgespielt.

Bei mehreren auditiven Agenten, die zeitgleich einen Hinweiston abspielen wollen, wird bis zum Ende vom Abspielvorgang des ersten Tones gewartet. Diese auditiven Agenten sind Objekte der Klasse AAFGenericAuditiveHintAgent und in der Lage über alle Objekte dieser Klasse zu iterieren, deren Abspielstatus zu überprüfen und gegebenenfalls für sich zu sperren (siehe Listing 8).

```
1  compute: aSamState
2  ...
3    noSoundIsPlaying := true.
4    temp := AAFGenericAuditiveHintAgent allInstances.
5    temp do:[
6      :i | i sound isPlaying
7        ifTrue:[
8          noSoundIsPlaying := false.
9          ].
10     ].
11     noSoundIsPlaying
12     ifTrue: [
13       sound play.          "spiele Ton ab."
14       myPart := self.      "setze Lock"
15       ].
16 ...
```

Listing 8: *Die Instanzmethode compute der auditiven Agenten.*

4.2.4 Erweiterung des Ereignissystems

Ein Abgrenzungskriterium ist die ausschließliche Benutzung der bereits entwickelten Agenten und nur bedingt Erweiterungen am Ereignissystem vorzunehmen. Ein anderes beinhaltet die ausschließliche Nutzung von auditiven und visuellen Buttons als Agenten-Buttons des AMD für die Automatisierung von Funktionen.

Da das AMD unter anderem Buttons für die Vergabe von visuellen und auditiven Geschwindigkeitshinweisen beinhaltet, musste das Ereignissystem erweitert werden. Ein Geschwindigkeitshinweis soll immer dann erscheinen, wenn die MWB das Tracking-Objekt schneller oder langsamer durch die Versuchsstrecke steuern sollen. Zwei ähnliche auslösende Ereignisse sind zwar vorzufinden („zu schnell fahren“ und „zu langsam fahren“), ihre Arbeitsweise ist allerdings nicht nachvollziehbar.

Im Zuge der Weiterentwicklung des Ereignissystem konnten zwei aktivierende Ereignisse („Strecke schnell fahren“ und „Strecke langsam fahren“) entwickelt werden, mit deren Hilfe Agenten in Automatiken integriert werden können, die die Aufgabe haben, die MWB selbstständig vor zu schnellem Fahren zu bewahren oder ein schnelleres Fahrverhalten zu fordern. Ihre Arbeitsweise wird am Ereignis „Strecke langsam fahren“[13] erläutert:

„Strecke langsam Fahren“:
Das neue Ereignis „Strecke langsam fahren“ soll die MWB vor zu schnellem Befahren der Teststrecke mit dem Tracking-Objekt bewahren und besteht aus zwei Zuständen. Der Agent, dem dieses Ereignis zugeteilt wurde, ist im Ausgangszustand inaktiv und wechselt beim Eintreffen von bestimmten Ereignissen in den aktiven Zustand. Da die Strecke nicht durchgehend gerade verläuft, sondern auch Kurven und Gabelungen beinhaltet, sind die optimalen Geschwindigkeiten jedes Streckenabschnittes dynamisch und individuell.

[13] die Arbeitsweise der beiden Ereignisauslöser ist analog

Der inaktive Zustand:
In diesem Zustand ist das auslösende Ereignis nicht eingetroffen und der Agent greift nicht ein. Dieser Zustand hat eine Aktivierungsgrenze, welche durch mehrere Kriterien festgelegt wird. Dies ist zum einen der Bereich zwischen 0.00 und 20.48, also der minimalen und maximalen „Geschwindigkeit“[14] des Tracking-Objekts. Zum anderen wird diese Grenze an der Funktion „optimalStep“[15] bemessen, die eine optimale „Geschwindigkeit“ des Tracking-Objekts, abhängig vom Streckenabschnitt und seiner relativen Lage zur Strecke, zurückgibt. Als Letztes fließt ein Toleranzwert ein, der im Aktivierungsmorph eingestellt werden kann (siehe Abbildung 11a). Er gibt an, wie konsequent das auslösende und das deaktivierende Ereignis eintreten soll. Sein Wertebereich bewegt sich zwischen null und zehn, wobei eine niedrige Zahl angibt, dass Agenten dieses Ereignisses gegenüber schnellem Fahrverhalten toleranter sind als beim Einstellen einer hohen Zahl. Das Ereignis wird also bei schnellerem Fahren später ausgelöst.

Jedes Ereignis muss die aus der abstrakten Oberklasse AAFEvent geerbte Instanzmethode `isValid` implementieren, welche einen boolschen Wert (also wahr oder falsch) zurückgibt und so Agenten, die mit diesem Ereignis verbunden sind, aktiviert oder sperrt. Der genaue Aufbau von Ereignissen kann in der Diplomarbeit von N. Kosjar[7] nachgelesen werden.

Das genaue Zusammenspiel der Werte beim Errechnen des Grenzwertes wird im Listing 9 im Detail dargestellt. Hier ist in den Zeilen fünf und zehn der Vergleich der aktuellen Geschwindigkeit `currentSpeed` mit der Summe aus der optimalen Geschwindigkeit `optimalSpeedWithoutTolerance`, dem Toleranzwert `toleranceBuffer` und dem Offset für das Festlegen der Grenzen zu sehen.

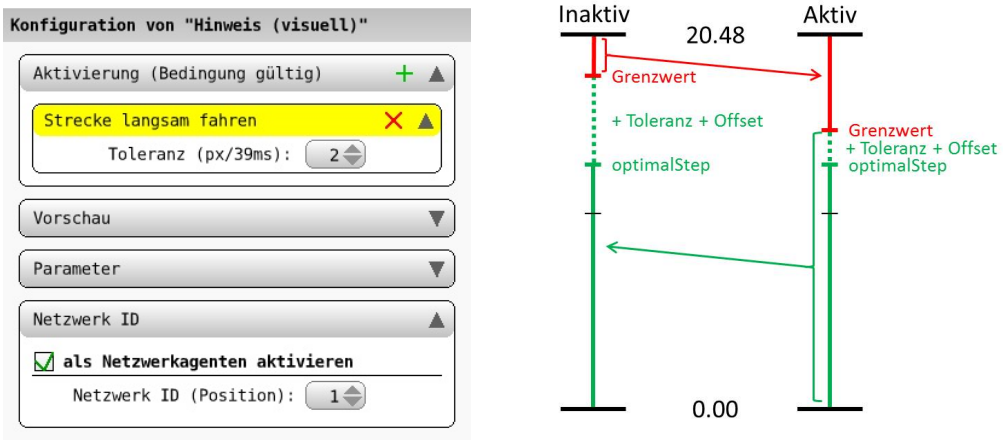
[14] als Geschwindigkeit des Tracking-Objekts wird die Anzahl der Pixelverschiebungen des Streckenbildes, abhängig von der/den Joystickauslenkung/en, beschrieben. Dies wirkt, als würde sich das Tracking-Objekt bewegen.
[15] entwickelt von H. Weidner-Kim[12]

```
isValid: aSamState
...
tooFast
  ifTrue:[        "Zustand 1: aktiviert"
    (currentSpeed < (optimalSpeedWithoutTolerance+(toleranceBuffer/5)+1))
    ifTrue:[
      tooFast := false.
    ]
  ]ifFalse:[      "Zustand 2: deaktiviert"
    (currentSpeed > (optimalSpeedWithoutTolerance+(toleranceBuffer/5)+3))
    ifTrue:[
      tooFast := true.
    ]
  ].
^tooFast
```

Listing 9: *Die Instanzmethode isValid des Ereignisses „Strecke langsam fahren“ mit den zwei Zuständen, die die Variable tooFast annehmen kann.*

Der aktive Zustand:
Überschreitet das Tracking-Objekt die Grenze der maximal errechneten „Geschwindigkeit“ für den jeweiligen Abschnitt, so wechselt das Ereignissystem in den Zustand „Aktiv“ (`tooFast = true`). Es wird also ein Hinweis in der SAM als Bild angezeigt oder als Ton abgespielt. Dieser Zustand wird verlassen, also in den inaktiven Zustand gewechselt, sobald die deaktivierende Grenze unterschritten wird, die wiederum von den gleichen Kriterien festgesetzt ist wie die im inaktiven Zustand (siehe Abbildung 11b).

Durch das Setzen von zwei Grenzen kann ein gewisser Aktivierungsbereich geschaffen werden. Dies ist gegenüber einer festen Grenze von Vorteil, weil zum Einen das Verhalten eines realen Operateurs nachgestellt wird, indem Hinweise über einen längeren Zeitraum angezeigt werden. Zum Anderen werden unerwünschte Effekte minimiert, wie die „blinkende Aktivierung“.



(a) Möglichkeit zur Einstellung des Toleranzbereiches

(b) Grenzen der Zustände; links: inaktiver Zustand; rechts: aktiver Zustand

Abbildung 11: *Event: „Strecke langsam fahren“ und Grenzen der Zustände*

Ein Beispiel für die „blinkende Aktivierung“:
Es wird angenommen, dass der Grenzwert für die Aktivierung und Deaktivierung auf einem konstanten Streckenabschnitt (z.B. bei einem geraden Streckenabschnitt) immer gleich ist. Da zwei MWB die Geschwindigkeit des Tracking-Objekts steuern ist davon auszugehen, dass dieser Grenzbereich innerhalb kürzester Zeit mehrmals überschritten werden kann. In dieser Zeit würden mehrere, sehr kurze Hinweise hintereinander angezeigt und die Aufgabe der Agenten verfehlt werden, da sie so die MWB nicht ausreichend unterstützen.

Mit einem Aktivierungsintervall wird ein gewisses zeitliches Anzeigen von Hinweisen und damit ein konsequentes Verhalten der Agenten sichergestellt. Somit können diese Hinweise also auch von den MWBn kognitiv abgearbeitet und der Operateur entlastet werden. Die Festlegung der Grenzen ist ausschließlich aus empirischen Daten gewonnen und so kalkuliert, dass bei der Benutzung beider Ereignisse gleichzeitig („Strecke schnell fahren“ und „Strecke langsam fahren“) ein Bereich existiert, in dem die MWB mit optimaler „Geschwindigkeit“ fahren, also keines der Ereignisse aktiv wird.

4.2.5 Logging

In der SAM-Auswertungsdatei werden zwei neue Spalten angelegt: eine für die Netzwerk-Button-Aktivitäten am AMD, die andere für die Netzwerk-Agenten-Aktivitäten des AAF eines Testdurchgangs. Inhaltlich sind diese zwei Spalten sinnvoll in die vorhandene Spalteneinteilung integriert. Sie wurden zwischen die Spalten der manuellen Operateurshinweise und den Spalten der Automatiken-Aktivitäten eingegliedert.

Das Loggen der Button-Aktivitäten:
Der Netzwerk-String, der vom Operateurs-Rechner über das Netzwerk den SAM-Rechner erreicht, enthält den Zustand über alle Agenten-Buttons des AMDs. Alle 39-40ms werden neue Daten ins Logfile geschrieben. Genau zu diesem Zeitpunkt durchläuft der komplette Netzwerk-String einen Algorithmus, der die aktiven Stellen aus dem String filtert, also nur die Stellen als durch Komma getrennte Integer zurückgibt, die aktiv sind. So wird kurze und verlustfreie Ausgabe erreicht, die die Bearbeitung der Versuchsdaten durch das LFA im Anschluss erleichtert. Bei Inaktivität aller Buttons wird der Wert „0“ ins Logfile geschrieben.

Das Loggen der Agenten-Aktivitäten:
Auch die Agenten-Aktivitäten werden in dieser Schreibweise im Logfile festgehalten. Allerdings lässt sich hier nicht einfach in aktiv und inaktiv unterscheiden. Den Unterschied stellen die visuellen Agenten dar, welche immer als Paar agieren. Der dafür entwickelte Algorithmus ist komplizierter, arbeitet jedoch fehlerfrei und effektiv. Hier werden doppelte Informationen gefiltert und bei Inaktivität aller Agenten der Automatik standardmäßig der Wert „0“ zurückgegeben und geloggt.

4.2.6 Start des neuen Modus

Als letzter Schritt der Anpassungen an SAM ist ein Morph zu nennen, der beim Starten des „Opermatik“-Assistenzmodus darauf hinweist, wenn keine Netzwerkverbindung aktiv ist (siehe Abbildung 12). Dies ist für die Kommunikation des Operateursarbeitsplatzes mit der SAM und den Automatiken des AAF dringend notwendig.

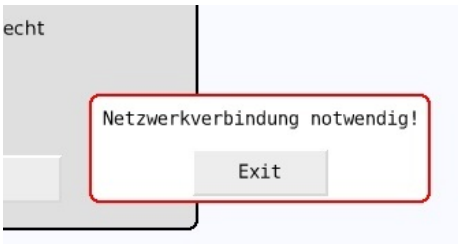


Abbildung 12: *SAM: Hinweis auf fehlende Netzwerkverbindung*

4.3 AMD

In diesem Unterkapitel werden die Anpassungen an das AMD beschrieben. Hierzu gehört nicht nur das Integrieren von Buttons, die eine neue Arbeitsweise aufweisen und damit mächtiger sind, als die bisherigen Buttons. Im Hintergrund mussten deren Aktivitäten mit Hilfe der Belegung von Button-Nummern in den Netzwerk-Verkehr integriert, eingehende Agenten-Aktivitäten ausgewertet und gegebenenfalls Hinweise angezeigt werden.
Als Letztes war es wichtig, dass vor jedem Streckenabschnitt ein Zurücksetzen der Button-Zustände und der Hinweise gewährleistet ist.

4.3.1 Entwurf der Anpassung des AMDs

Die offensichtlichste und für den Nutzer einzig sichtbare Überarbeitung an dem AMD-GUI sind die neuen Agenten-Buttons. Ältere Buttons führen eine Instanzmethode (sogenannte *ActionSelector*-Methoden) aus, wenn mittels der rechten oder linken Maustaste auf sie geklickt wird. Die neuen LR[16]-Buttons differenzieren zwischen der Anwahl mit der linken oder der rechten Maustaste. Hierfür musste eine Button-Klasse (OPButtonWithMouseAction) geschaffen werden, mit welcher genau diese Arbeitsweise machbar ist. Mit dieser Klasse kann der neue Agenten-Button universell in einer Instanz- oder Klassenmethode eingebunden werden, wobei direkt zwei *ActionSelector*-Methoden als Parameter angegeben werden.

Die Funktionen des neuen Buttons:
Während ältere Buttons bei Aktivierung eine Aktion ausführten, kann beim neuen Button-Typ mit der rechten Maustaste ein Zustand gehalten werden. Das ist notwendig, um Agenten über einen gewünschten Zeitraum aktiv zu halten. Dabei wird ein gelber Rahmen um den Agenten-Button gelegt (siehe Abbildung 13), der anzeigt, dass Agenten hinter dem Button in der SAM

[16] LR = links, rechts als Hinweis auf die neue Differenzierung bei Anwahl des Buttons mit der linken und rechten Maustaste

aktiv sind und gegebenenfalls eingreifen können. Ist dies der Fall, also kann ein Agent das SAM-Abbild manipulieren, dann kann ihn der Operateur jederzeit deaktivieren. Entweder durch das erneute Drücken der Schaltfläche mit dem rechten oder mit Drücken der linken Maustaste[17]. In letzterem Fall werden zwei Aktionen gleichzeitig ausgeführt. Zum einen wird der Agent deaktiviert, zum anderen wird umgehend ein Operateurshinweis abgesendet und bei den MWBn eingeblendet, also sofort reagiert, wie bei der manuellen Vergabe von Hinweisen des Operateurs.

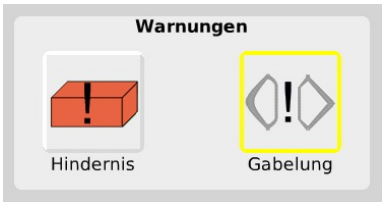


Abbildung 13: *AMD links: inaktiver Button; rechts: aktiver Button*

Die Belegung der neuen Buttons mit Nummern:
Für die Aktivierung von einzelnen Agenten einer Automatik des AAF mussten die Buttons mit Nummern versehen werden, die synchron zur Netzwerk-ID der Agenten ist (siehe Tabelle 4). Die dazugehörigen Button-Nummern sind wie folgt im AMD verteilt:

[17] z.B. in Notsituationen, in denen der Operateur dem Agenten nicht vertraut und eingreifen will

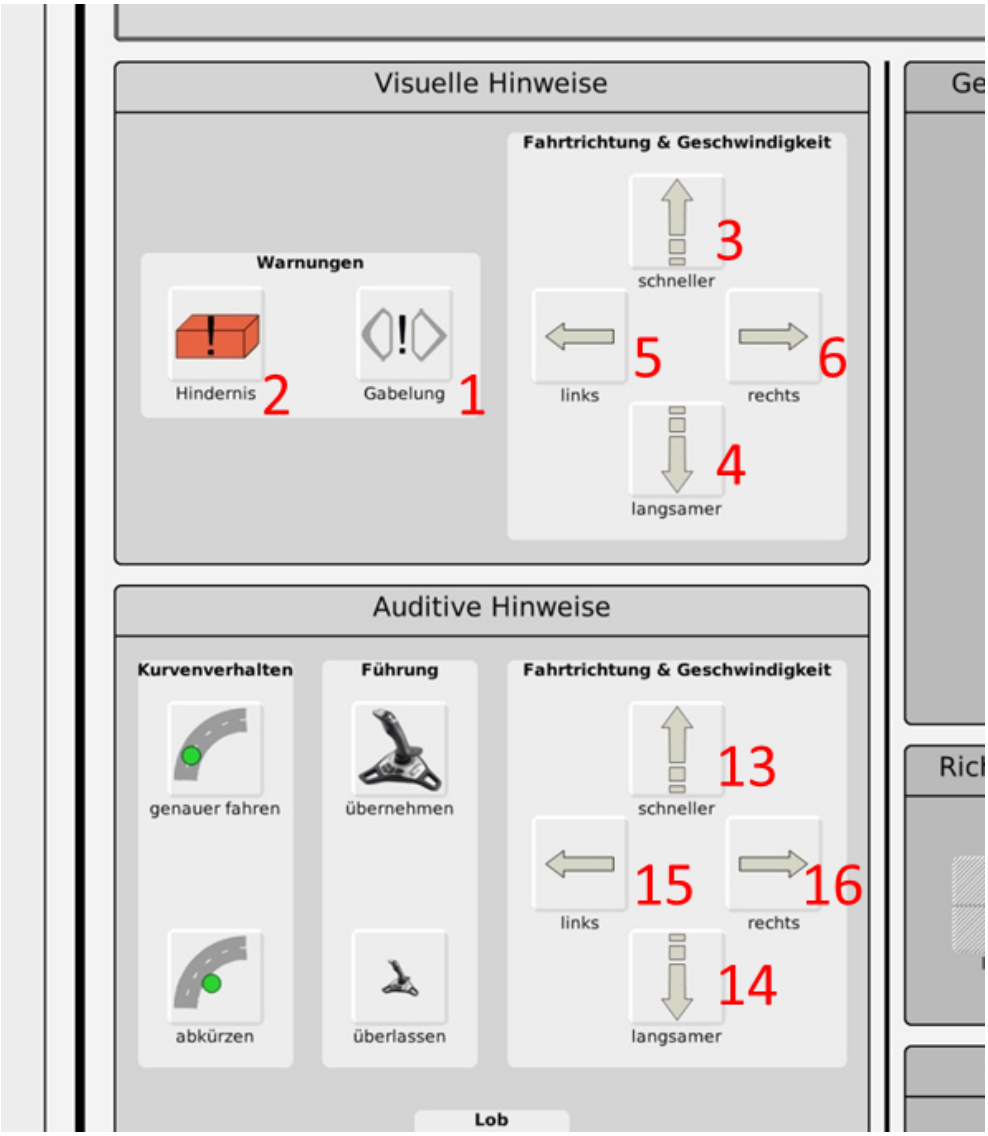


Abbildung 14: *AMD: Vergabe der Button-Nummern (rot) im „Visuelle Hinweise“- und „Auditive Hinweise“-Morph.*

Diese Nummern sind in der `initialize` Methode der Klasse OPAbstract-Automatic festgeschrieben (siehe Listing 10) und können an dieser Stelle beliebig geändert werden. Sie umfassen, genau wie die Netzwerk-IDs und deren Netzwerk-String, dreißig Stellen und sind nach visuellen (1-10), auditiven (11-20) und sonstigen (21-30) Agenten-Funktionen unterteilt.

```
1  initialize
2  ...
3    strgPosition := Dictionary new.
4    strgPosition at: #forward put: 3;
5              at: #left put:  5;
6              at: #right put: 6;
7              at: #back put: 4;
8              at: #obstacle put: 2;
9              at: #fork put: 1;
10             at: #forwardAudio put: 13;
11             at: #leftAudio put:  15;
12             at: #rightAudio put: 16;
13             at: #backAudio put: 14.
14 ...
```

Listing 10: *Die initialize Methode der Klasse OPAbstractAutomatic*

Zurücksetzen bei Start einer neuen Versuchsstrecke:
Beim Testdurchgang des neuen Modus ist es notwendig, dass alle der derzeit elf Streckenabschnitte unter gleichen Bedingungen beginnen. Im Einzelnen bedeutet das, dass angezeigte Hinweise und aktive Agenten-Buttons aus dem vorherigen Durchgang zurückgesetzt (engl. reset) werden müssen. Dies wird in den Instanzmethoden `resetHintsForNewStep` und `resetButtonsForNewStep` der Klasse OPAbstractAutomatic realisiert (siehe Listings 11 und 12).

In der Methode `resetButtonsForNewStep` wird der ausgehende Netzwerk-String zurückgesetzt[18], über alle Agenten-Buttons iteriert und deren integrierte reset-Methode aufgerufen, welche den Ausgangszustand der Buttons wiederherstellt.

[18] also an allen dreißig Stellen des Netzwerk-Strings der Wert „0“ eingetragen

```
resetButtonsForNewStep
|temp|
  modelExport setAutomationCodeZeros.
  temp := OPButtonWithMouseAction allInstances.
  temp do: [
    :i | i resetButton.
  ].
  self resetHintsForNewStep.
```

Listing 11: *AMD: Zurücksetzen der Buttons*

In der Methode `resetHintsForNewStep` dagegen werden alle Agenten-Hinweise des AMD ausgeblendet und Anzeigesperren aufgehoben.

```
resetHintsForNewStep

  lock := 0.
  shownHint ~= nil
  ifTrue:[
      (leftGfx at: shownHint) hide.
      (rightGfx at: shownHint) hide.
    ].
```

Listing 12: *AMD: Zurücksetzen der Agenten-Hinweise*

4.3.2 Implementierung der Schnittstellen

Das Aktivieren eines Buttons mittels der rechten Maustaste veranlasst, wie zuvor beschrieben, die Änderung der Farbe des Button-Rahmens. Dies ist eine grundlegende Funktion des Buttons selbst und nicht der Instanzmethode, welche im gleichen Zuge durchlaufen wird. Alle Instanzmethoden der neuen Buttons senden ein eigenes, eindeutiges Signal an die Methode `setNewCode` der Klasse OPAbstractAutomatic. Diese Methode kann dem Signal eine Zahl zuordnen, die die Position im Netzwerk-String darstellt und manipuliert diesen im Anschluss an gewünschter Stelle. Der String wird dann im Netzwerk-Paket zum SAM-Rechner gesendet und ausgewertet. In diesem Schritt wird also festgelegt, ob ein Agent der Automatik aktiv ist und gegebenenfalls eingreifen darf, oder nicht.

Greift ein Agent in den Trackingprozess ein, wird also seine `compute`-Methode im AAF ausgeführt, so erhält der AMD-Rechner diese Informationen durch den im Netzwerk-Paket enthaltenen Netzwerk-String und hat dann die Aufgabe, dem Operateur am AMD eine Statusmeldung zurückzugeben. Das geschieht in gleicher Form, wie die Statusanzeige der Hinweise, die der Operateur manuell erteilen kann (siehe Abbildung 3). Dabei stellte sich die Implementierung der Schnittstelle als komplexer heraus, als die der Schnittstelle AMD → SAM, bei welcher mehrere Agenten gleichzeitig aktiv sein können. Das AMD schreibt die Anzeige von aktuell angezeigten oder abgespielten Hinweisen links und rechts der SAM-Streckenansicht vor, wobei immer nur ein Hinweis angezeigt werden kann. Dadurch gehen Informationen verloren, wenn mehrere Agenten gleichzeitig eingreifen. Für diesen Fall wurden zwei Lösungen entwickelt und integriert:

Die Anzeige von manuellen Hinweisen des Operateurs:
Greift ein Agent in den Trackingvorgang ein und zeigt z.B. einen visuellen Hinweis in der SAM an, so wird diese Aktion auch am AMD angezeigt. Es ist allerdings dem Operateur möglich, im gleichen Moment einen manuellen Hinweis zu geben. Diese wurden so programmiert, dass sie sich dominant gegenüber den Agenten-Hinweisen verhalten. In diesem Fall wird also der Agent unterbrochen; das Anzeigen des Operateurshinweises dominiert. Somit kann sichergestellt werden, dass der Operateur zu jedem Zeitpunkt in der Lage ist, aktiv in den Prozess einzugreifen, ähnlich wie beim Deaktivieren des Agenten-Zustands der Buttons. Endet der Operateurshinweis, so werden die Aktionen der Agenten am AMD und in der SAM wieder sichtbar.

Die Anzeige bei mehreren aktiven Agenten-Hinweisen:
Beim gleichzeitigen Eingreifen von mindestens zwei Agenten muss ebenso ein Dominanzverhalten wie bei Operateurshinweisen integriert werden. Für genau diesen Zweck konnte ein Algorithmus entwickelt werden, mit dem es möglich ist, die Anzeige von Agenten-Aktionen für andere Agenten zu sperren (engl. lock). Der Ablauf des Algorithmus ist denkbar einfach: Alle 39-40ms wird untersucht, ob Agenten aktiv sind und bereits ein Lock gesetzt ist.

Ist dies nicht der Fall oder wird der Lock durch einen Agenten selbst gehalten, so wird der Agenten-Hinweis im AMD angezeigt (siehe Abbildung 15). Es ist kein Lock aktiv, wenn z.B. ein neuer Streckenabschnitt angefangen hat oder ein Agent seinen Lock aufgibt, weil er nicht mehr in der SAM eingreift.

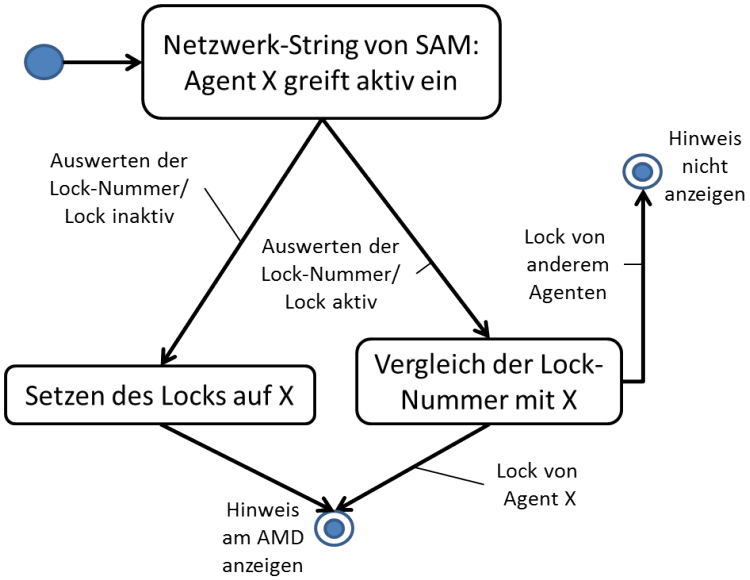


Abbildung 15: *AMD: Locking-Algorithmus für die Anzeige von Agenten-Hinweisen am AMD.*

4.4 LFA

Das LFA ist ein in der Programmiersprache Java geschriebenes Werkzeug zur Auswertung von ATEO-Logfiles. Sein GUI ermöglicht dem Benutzer, die gespeicherten Versuchsdaten einzulesen und den Auswertungsvorgang zu starten. Das Ergebnis ist ein strukturierter Ordner, der eine Microsoft Excel-Datei enthält, in welcher mehrere Tabellen in verschiedenen Registern[19] vorzufinden sind sowie mehrere Charts zur Auswertung der Eingriffe in der SAM. Als weitere Erweiterung des LFA ist das automatische Erstellen eines „results"-Ordners zu nennen, in welchem diese Daten nach dem Ausführen des LFA zu finden sind. In der früheren Version des LFA musste dieser manuell erstellt werden, da es andernfalls nicht korrekt gestartet werden konnte.

4.4.1 Erweiterung des Auswertungsfiles

Durch den neuen Testmodus und die Überarbeitung aller Frameworks des ATEO-Systems sind neue Daten angefallen, welche verlustlos und komprimiert in die Auswertungstabelle integriert werden. Für diesen Zweck sind zwei neue Tabellenblätter und Register entstanden:

Das Tabellenblatt „OP-ButtonsUndAutomatiken":
Die Kopfzeile der Tabelle besteht aus einer Stepnummer, also der Nummer des Streckenabschnittes, aus einer Zeitmarke und der Button- und Automatik-Aktivität. Die Auswertung der Streckenabschnitte erfolgt geordnet nach Nummer der einzelnen Abschnitte, wobei jede Zeile der Tabelle für mindestens eine Änderung der Button- und/oder Automatik-Aktivität gesehen wird. Welche Aktivitäten zu einer bestimmten Zeitmarke genau eingetroffen sind, kann durch den Vergleich zur darüberliegenden Zeile abgelesen werden (siehe Abbildung 16).

[19] Register sind kleine Schaltflächen in Microsoft Excel, mit denen man zwischen den verschiedenen Tabellenblättern auswählen kann.

	A	B	C	D	E
1	StepNummer	Zeitmarke	Buttonaktivitaet	Automatikaktivitaet	
2					
3	1	13582	3	0	
4	1	14719	3	3	
5	1	15424	3	0	
6	1	16406	0	0	
7	1	19192	3	0	
8	1	23791	3	3	
9	1	24537	3	0	
10	1	28226	3	0	
11	2	20883	0	0	
12	3	16569	2	0	
13	3	17078	1,2	0	
14	3	17549	1,2,5	0	
15	3	17943	1,2,5,6	0	
16	3	18337	1,2,3,5,6	0	
17	3	18850	1,2,3,4,5,6	0	
18	3	19086	1,2,3,4,5,6	4	
19	3	20343	1,2,3,5,6	0	
20	3	25009	1,2,3,5,6	0	
21	4	6985	15	0	
22	4	7839	15	15	
23	4	8977	15	0	
24	4	10489	15	15	
25	4	11471	15	0	

Abbildung 16: *LFA: Tabellenblatt OP-ButtonsUndAutomatiken mit StepNummer, Zeitmarke, Button- und Automatik-Aktivität. Jede Zeile steht für mindestens eine Aktivität des Buttons und/oder der Automatik zu einer Zeitmarke oder für die Endzeitmarke des Versuchsabschnitts.*

Das Tabellenblatt „OP-BA-Zusammenfassung“:
Dieses Tabellenblatt ist eine Zusammenfassung der Button- und Automatik-Aktivitäten für jede einzelne Automatik jedes Versuchsabschnittes.
Im Gegensatz zum ersten Tabellenblatt wird hier keine Liste der Abfolge der einzelnen Aktivitäten erstellt, sondern eine dynamische Liste von Tabellen (siehe Abbildung 17) mit Hilfe eines komplexen Algorithmus. Für jeden Versuchsabschnitt werden alle Automatiken als kleine Tabelle sortiert nebeneinander (von links nach rechts) in die Tabelle geschrieben, wenn sie durch den Operateur am AMD mit dem jeweiligen Button aktiviert wurden oder als Automatik eingegriffen haben. Dieser Vorgang wird für alle Versuchsabschnitte durchlaufen (geordnet von oben nach unten). Alle inaktiven Automatiken und Buttons werden nicht in das Tabellenblatt geschrieben, wodurch die Übersichtlichkeit bei gleichbleibendem Informationsgehalt erhöht werden konnte.

	A	B	C	D	E	F	G	H
1	Step: 1 - Automatik: 3							
2	Button aktiv von	bis	Automatik aktiv von	bis				
3	13582	16406	14719	15424				
4	19192	28226	23791	24537				
5								
6								
7	Step: 3 - Automatik: 1					Step: 3 - Automatik: 2		
8	Button aktiv von	bis	Automatik aktiv von	bis		Button aktiv von	bis	Automatik aktiv von
9	17078	25009				16569	25009	
10								
11								
12	Step: 4 - Automatik: 15							
13	Button aktiv von	bis	Automatik aktiv von	bis				
14	6985	12257	7839	8977				
15		24615	10489	11471				
16								
17								
18	Step: 7 - Automatik: 1					Step: 7 - Automatik: 2		
19	Button aktiv von	bis	Automatik aktiv von	bis		Button aktiv von	bis	Automatik aktiv von
20	33485	97355	41469	43219		33014	97355	401
21			57109	59249				611
22			63455	65161				

Abbildung 17: *LFA: Tabellenblatt OP-BA-Zusammenfassung mit dynamischer Erstellung von zusammenfassenden Tabellenausgaben der einzelnen Agenten der Automatik (von links nach rechts) in jedem Versuchsdurchgang (von oben nach unten)*

4.4.2 Grafische Darstellung der Agenten-Aktivitäten

Um die Häufigkeit des Auftretens eines Agenten und aller Agenten innerhalb der Automatik pro Versuchsabschnitt besser einschätzen zu können, wurde eine visuelle Darstellung der Aktivitäten entwickelt (siehe Abbildung 18).

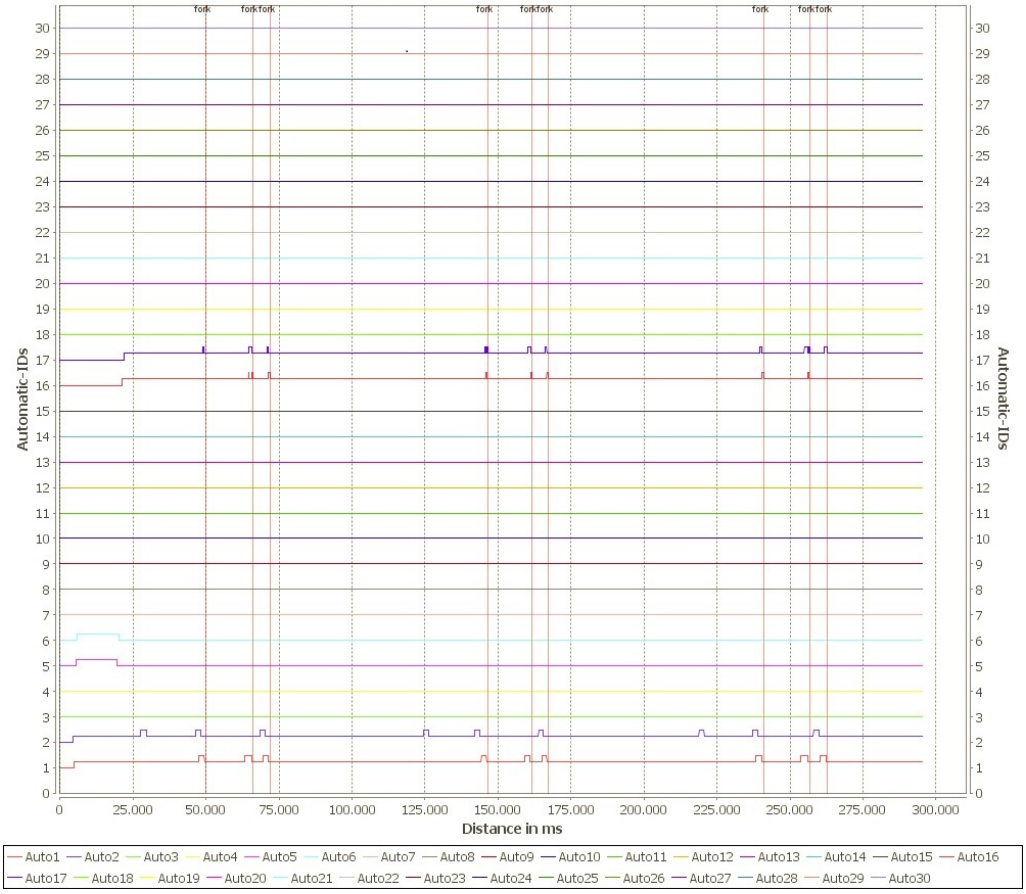


Abbildung 18: *LFA: grafische Darstellung der Agenten-Aktivitäten mit Erhebungen im Graphen bei Aktivität des Buttons und weiterer Erhebung bei Agenten-Eingriffen.*

Diese Darstellung enthält dreißig Graphen (ein Graph steht für jede Agenten-Nummer (ID) innerhalb der Automatik), welche horizontal im Koordinatensystem verlaufen. Aktiviert der Operateur einen Button am AMD, so erhebt sich der Graph des Buttons in dieser Zeitperiode um 0,25.

Ist in der Periode des aktiven Buttons ein Eingreifen des entsprechenden Agenten erfolgt, so erhebt sich der Graph um insgesamt 0,5. Dabei spiegelt die Länge des Graphen die Länge des Versuchsabschnittes in Millisekunden auf der x-Achse dar. Durch die Wahl einer stufenförmigen Darstellung der Aktivitäten können alle Button- und Agenten-Aktionen geordnet, übersichtlich und verlustlos präsentiert werden. Zusätzlich ist das Auftreten von Gabelungen als vertikale rote Linie an entsprechender Zeitmarke im Koordinatensystem integriert worden. Das Filtern der genauen Zeitmarken der Gabelungen wurde nicht an den festen Koordinaten vorgenommen, sondern an den Farbcodes der Streckensegmente[20], der die Gabelungen enthält. Damit ist eine fehlerfreie Arbeitsweise des LFA auch nach der Änderung von Versuchsstrecken gewährleistet.

[20]Diese Farbcodes, auch Colorcodes genannt, sind kleine farbliche Streifen an der linken Seite jedes Teilabschnittes und für diese eindeutig.

4.5 Tests

In diesem Abschnitt werden die verschiedenen Arten und Umfänge der Tests über das gesamte ATEO-System nach der Integration des neuen Versuchsmodus vorgestellt. Zuerst werden die Tests der neuen Ereignisse und anschließend der Versuchsablaufstest, der die einzelnen Module, Button- sowie Automatik-Funktionen beinhaltet und in verschiedenen Minimal-, Normal- und Extremkonstellationen testet, näher beschrieben. Danach folgt der Ablauf der Auswertungstests am Beispiel mehrerer unterschiedlicher Logfiles und Testvarianten.

Die Integration der Testumgebung, welche parallel zu dieser Diplomarbeit von N. Kosjar[7] entwickelt wird oder der Testverfahren, die von Smalltalk gegeben sind, ist nur teilweise anwendbar. Bei diesen Testvarianten können kleinere Bestandteile als Code-Segmente oder Ereignisse des Ereignissystems getestet werden aber nicht größere, zusammenhängende Frameworks oder die Verwendung von GUIs. In dieser Arbeit wurden alle denkbaren Varianten eines Versuchsablaufes manuell durch den Entwickler und die Betreuerin (C. von Bernsdorff) getestet und begutachtet.

4.5.1 Tests der neuen Ereignisse

Für die Umsetzung der Anforderungen war es notwendig zwei neue Ereignisse für das Ereignissystem zu entwickeln (siehe 4.2.4). Um diese zu testen, wurde die Testumgebung von N. Kosjar[7] verwendet, welches auf SUnit-Tests basiert und eine visuelle Anzeige bietet, die in Abbildung 19 zu sehen ist.

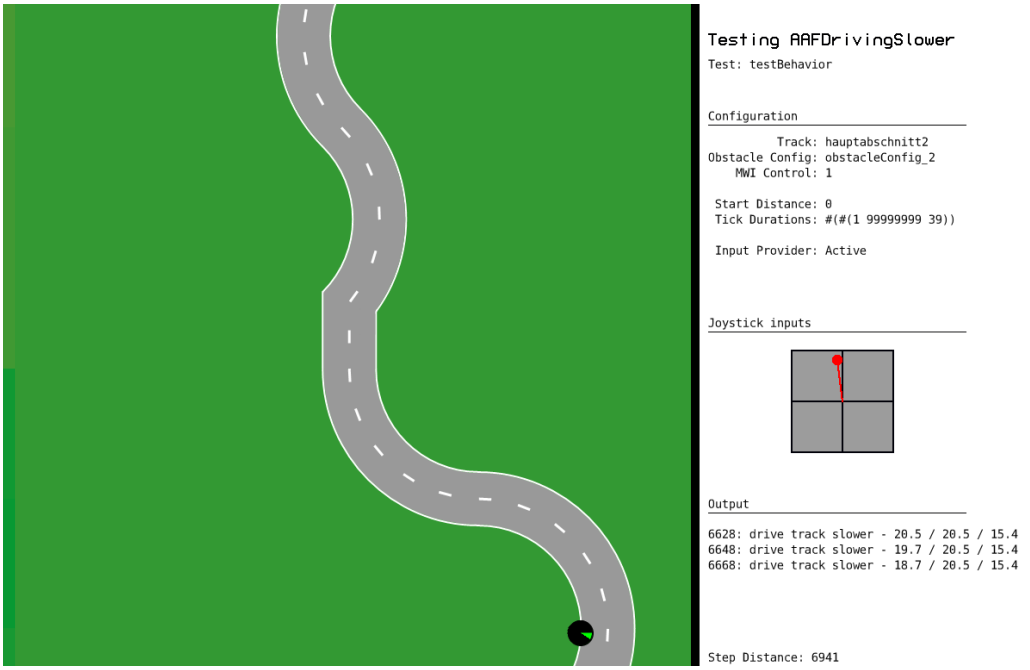


Abbildung 19: *Tests: grafische Anzeige des Testdurchgangs eines Ereignisses durch die Testumgebung*

Für das Durchführen der Test wurden zwei neue Test-Klassen angelegt (`AAFDrivingSlowerTest` und `AAFDrivingFasterTest`), welche Instanzmethoden integrieren, die die für den Test notwendigen Parameter beinhalten. Getestet wurde das Verhalten der Ereignisse auf den ersten 15000 Pixeln der Strecke „Hauptabschnitt2“. Diese Teilstrecke enthält alle möglichen Kurven der ATEO-Versuchsstrecken. Durch die dynamische Berechnung der optimalen Geschwindigkeiten sind Kurven als besonders kritische Streckenabschnitte zu sehen. Die weiteren Parameter sind im Listing 13 und die dazugehörigen Erklärungen sowie der Testablauf in der Diplomarbeit von N. Kosjar [7] zu finden.

```
testBehavior
  "self debug: #testBehavior"

  | event |

  "CONFIGURE EVENT"
  event := AAFDrivingSlower new.
  event toleranceBuffer: 8.

  "SETUP AGENT"
  self setupAgents:
    (Array
      with: (AAFInputProviderAgent withJoystickInputs: (self class
          testInput))
      with: (AAFTestAgent withDelegate: event)
    ).

  "SETUP STEP"
  self configureStepMwiControl: '1'.
  self configureStepTrack: 'hauptabschnitt2'.
  self configureStepObstacleConfig: 'obstacleConfig_2'.
  self configureStepStartAtDistance: 0."8500."
  self configureStepEndAtDistance: 15000.
  self configureStepTickDuration: #(
    "#(0 99999999 5)"#(1 99999999 39)
  ).

  "RUN STEP"
  self runStep.

  "PRINT & COMPARE OUTPUT"
  self printAndCompareOutputWith: self class testOutputReference.
```

Listing 13: *AMD: Methode testBehavior der Testklassen für die beiden neuen Ereignisse*

Um variierende dynamische Joystick-Eingaben zu simulieren, wurden für diesen Versuch eigene Eingaben verwendet werden, die vorher erzeugt wurden. Diese sind in der Klassenmethode `testInput` der beiden Test-Klassen zu finden. Die Referenzdaten für den Vergleich der Daten der Testdurchgänge befinden sich in der Klassenmethode `testOutputReference`.
Auf diese Weise wurden beide Ereignisse mehrmals getestet. Es traten keine Fehler auf.

4.5.2 Tests der Versuchsabläufe

Die Art des Versuchsablaufes variiert je nach Versuchsmodus. Das heißt, dass die MWB entweder durch eine Automatik, durch einen Operateur oder durch einen Operateur mit Automatiken überwacht und unterstützt werden können. Mit der Integration des letzen Modus dürfen die bisherigen Modi nicht beeinträchtigt oder in ihrer Arbeitsweise gestört werden.

Test der Automatiken:
Die übrigen Agenten-Ereignisse der Automatiken, die in dieser Arbeit Verwendung fanden, sind als getestet angenommen worden. Sie entstanden im Zuge der Entwicklung des Ereignissystems und der visuellen und auditiven Agenten in der Diplomarbeit von N. Kosjar[7].

Test MWB und Automatik:
Der Test des Automatik-Modus wurde mit variierenden Automatiken über die komplette Distanz oder über Teilabschnitte durchgeführt. Dies garantiert anschließend auch die Auswertung der Daten von unkompletten Testdurchläufen. Dabei wurde nicht nur die Automatik „OPAutomatik“ (siehe 4.2.3) verwendet, sondern auch die Automatik „Konzept36“[21] und die Dummy-Automatik, welche keine Agenten enthält. Die Quelle ist also direkt mit der Senke verbunden, und die Daten werden ohne Manipulierungen durchgereicht. Sie ist also eine Automatik ohne Agenten und greift nicht in den Prozess der SAM ein.

[21] Die Automatik „Konzept36“ ist ein Zusammenschluss mehrerer Agenten, die im Rahmen einer Studie von Saskia Kain im ATEO-Projekt durch Entwickler aus Industrie und Forschung zusammengestellt wurden. Diese sollten eine Automatik konzipieren, die den Operateur ersetzen und die MWB allein unterstützen könnte. Die Automatik des Teams 36 wurde dabei als eine der Besten der 30 Teams dieser Studie ermittelt.

Beim Testen der „OPAutomatik“ konnte festgestellt werden, dass im Ereignissystem von N. Kosjar[7] nicht mehr als drei auditive Agenten in der Automatik integriert sein dürfen. Bei mehr als drei Agenten ist die virtuelle Maschine von Squeak durchgehend abgestürzt. Um den Fehler zu beheben, konnten im Zuge dieser Diplomarbeit die mp3-Musikdateien in das wav-Musikformat umgewandelt und die entsprechenden Einlesemethoden geändert werden. Der Auslöser für das fehlerhafte Verhalten war das Einlesen von mp3-Dateien als digitaler Datenstrom (engl. stream) in Squeak, welcher nur begrenzte Speicherreserven zur Verfügung hat und andererseits das lückenhafte Testen des Ereignissystems vom Entwickler. Durch das unterschiedliche Einlesen und Abarbeiten der wav- und mp3-Musikformate in Squeak musste im letzten Schritt die Lautstärke der wav-Tondateien um 50% angehoben werden, um sie dem Lautstärkeniveau der mp3-Töne anzugleichen. Alle Tests konnten anschließend erfolgreich mit wav-Dateien durchgeführt und abgeschlossen werden.

Test MWB und Operateur:
Erfolgt die Überwachung der MWB allein durch den Operateur, so wird nicht der OA in der Version 3.0 verwendet, sondern sein Vorgänger in der Version 2.8.4. Der Grund dafür sind die speziellen Agenten-Buttons des neuen Operateursarbeitsplatzes, welche im Operateur-Modus nicht benötigt werden. Die restlichen Funktionen und die visuelle Gestaltung sind in beiden Varianten analog und zeigten nach mehreren Komplettdurchläufen mit der angepassten SAM keine Auffälligkeiten oder Fehler.

Test MWB und „Opermatik“:
Durch die Entwicklung des neuen Modus als Zusammenschluss der beiden vorherigen Modi, war ein umfassendes Testen unumgänglich. Einzelne Teiltests konnten dabei bereits in der Entwicklung an Prototypen abgeschlossen werden. Das ausführliche Testen des Gesamtresultats erfolgte durch verschiedene Testvarianten:

- Der Test des „Opermatik“-Modus wurde mit Hilfe der Dummy-Automatik erfolgreich getestet. Diese Art von Automatik ist dringend erforderlich, weil die ersten acht der insgesamt elf Streckenabschnitte ohne die Unterstützung durch eine Automatik absolviert werden sollen. Bei diesem Test sind keine Fehler aufgetreten.

- Der Test des „Opermatik“-Modus konnte anschließend mit der Automatik „OPAutomatik“ getestet werden. Dabei wurde die Automatik nicht nur für die letzten drei Streckenabschnitte eingesetzt, sondern auch im ganzen Testdurchlauf. Alle Tests verliefen mit positivem Ergebnis.

- Als Letztes wurde der „Opermatik“-Modus mit einer Automatik getestet, welche teilweise Netzwerk-Agenten und Nicht-Netzwert-Agenten beinhaltete. Der resultierende Effekt ist, dass die MWB durch die Weiterentwicklungen des Projektes durchgehend von Nicht-Netzwerk-Agenten[22] und partiell von Agenten, die vom Operateur angesteuert werden, unterstützt werden können.

4.5.3 Tests des Auswertungsablaufes

Durch die unterschiedlichen Testdurchgänge konnten viele verschiedene Auswertungsdaten gewonnen werden, welche nicht nur in der entstandenen CSV-Datei manuell eingesehen, sondern auch mit dem LFA ausgewertet wurden. Dabei entstanden folgende Resultate:

- Alle Auswertungsdaten des Automatik-Testmodus konnten analog zur LFA-Version 1.9 ausgewertet werden. Das Tabellenblatt „OPButtonsUndAutomatiken“ zeigt ausschließlich die Endzeitmarke jedes Streckenabschnittes an und keine Netzwerk-Button- oder Netzwerk-Automatik-Aktivität. Das Tabellenblatt „OP-BA-Zusammenfassung“ bleibt durch seinen dynamischen Tabellenaufbau leer.

[22] z.B. durch eine Streckenvorschau, die für die MWB durchgehend sichtbar ist.

- Alle Daten des Operateur-Testmodus weisen die gleichen Auswertungsdaten wie der Modus Automatik auf.
- Alle Daten des Testmodus „Opermatik“ konnten nach dem Auswerten durch das LFA mit den Aufzeichnungen während der Tests verglichen und als fehlerfrei befunden werden, genauso wie auch die Erstellung der einzelnen OACharts. Das Integrieren der Gabelungen in die Charts wurde anhand von Farbcodes der jeweiligen Teilabschnitte realisiert, ist damit dynamisch und war ebenso fehlerfrei.
- Beim Testen von unvollständigen Datensätzen durch das LFA sind keine Auffälligkeiten im Auswertungsvorgang aufgetreten. Hierbei muss lediglich erwähnt werden, dass die XML-Versuchsablaufsdatei, die durch das SAM-Framework erstellt wird, nicht vollständig ist, weil diese erst nach Abschluss des letzten Streckenabschnittes komplettiert wird. Die XML-Dateien mussten am Dateiende manuell um die Zeilen `</stepInfo>` und `</experiment>` ergänzt werden. Die anschließenden Auswertungen durch das LFA verliefen ohne Probleme.

5 Zusammenfassung und Ausblick

In diesem Kapitel wird die Umsetzung und der Umfang dieser Diplomarbeit zusammengefasst und anschließend ausgewertet. Dabei wird auch auf Probleme, die bei der Programmierarbeit aufgetreten sind, eingegangen und einige Ideen und der Ausblick für zukünftige Entwicklungen präsentiert werden.

5.1 Zusammenfassung

Ziel der Diplomarbeit war die Entwicklung eines dritten Testmodus, in dem der Operateur des ATEO-Systems die Möglichkeit hat, die einzelnen Agenten einer Automatik vom AMD aus anzusteuern. Hierfür konnte das AMD-Framework überarbeitet werden, sodass einige Buttons um neue Funktionen ergänzt wurden und das Automatiken-GUI des AAF Agenten als Netzwerk-Agenten deklarieren kann. Diese Netzwerk-Agenten können dann im Versuch vom Operateur mit Hilfe der Buttons aktiviert werden. Für die anschließende Auswertung der Logfiles wurde das LFA erweitert, ohne das Auswerten älterer Testdaten zu beeinflussen. Seine Benutzung erfordert keine Programmierkenntnisse der Sprachen Smalltalk oder Java.

Im Rahmen dieser Arbeit wurde zunächst das bestehende ATEO-System umfassend analysiert, um die Anforderungen festzulegen. Dabei konnte ermittelt werden, dass die enthaltenen Komponenten, wie z.B. der die Programmiersprachen Smalltalk und Java, oder das XML- und CSV-Datenformat der Auswertungsdateien, weiterhin verwendet werden konnten. Nach der Klassifizierung der ATEO-Komponenten in Frameworks, konnten die Arbeiten mit Framework-Implementierungen vorgenommen werden.

Die sichtbaren Neuerungen sind die Differenzierung der Mausanwahlen von neuen Schaltflächen des AMD in linke und rechte Maustaste mit jeweils unterschiedlichen Funktionen. Die linke Maustaste führt die gewohnten manuellen Eingriffe durch den Operateur aus. Die rechte Maustaste aktiviert und deaktiviert Agenten einer Automatik über das Netzwerk. Diese Agenten übernehmen dann die Aufgaben, die der Operateur mit Drücken des Buttons abarbeitet. Das SAM-Konfigurations-GUI wurde um den „Opermatik"-Assistenzmodus ergänzt. Als Letztes konnte der Konfigurationsbereich des AAF-Automatiken-GUIs um die Option „Netzwerk ID", mit welcher jeder Agent als Netzwerk-Agent mit entsprechender Button-Nummer versehen werden kann, und zwei neue auslösende Ereignisse erweitert werden.

Für die Auswertung der erweiterten Versuchsdatensätze wurde das LFA so editiert, dass der Anwender keine Änderungen bei dessen Benutzung, sondern nur im Resultat des Analysevorgangs vorfindet. Neben zwei zusätzlichen Tabellenblättern in der entstandenen Microsoft Excel-Datei, welche die zeitliche Änderung von Button- und Automatik-Aktivitäten sowie zusammenfassende Tabellen enthalten, entstehen Bilddateien. Sie beinhalten ein Koordinatensystem, in welchem alle Aktivitäten jedes Versuchsabschnittes übersichtlich präsentiert werden.

Die Komplexität der Diplomarbeit ergibt sich also aus dem Editieren und Zusammenbringen aller Komponenten des ATEO-Systems zu einem neuen Testmodus, der Weiterentwicklung von Ereignissen für Agenten und der Auswertung der so entstandenen Datensätze.

5.2 Probleme und Ideen

5.2.1 Die Integration von auditiven Agenten

Die Integration von mehr als drei auditiven Agenten in einer Automatik mit dem bestehenden Ereignissystem von N. Kosjar[7] brachte einige Probleme mit sich. Das Einlesen der mp3-Dateien als digitaler Datenstrom führte beim Starten des ATEO-Systems zu schwerwiegenden Fehlern, die schließlich im Abstürzen der virtuellen Maschine resultierten. Hier wurde auf die Benutzung der Ton-Dateien im wav-Audioformat zurückgegriffen und die entsprechenden Einlesemethoden umprogrammiert.

5.2.2 Das Zusammenstellen von Automatiken

Das Zusammenstellen von Agenten zu einer Netzwerk-Automatik im Automatiken-GUI ist nicht aufwendig. Ein Hindernis ergibt sich jedoch: da die Agenten die gleiche Netzwerk-Nummer haben müssen, wie die Buttons am AMD, mit denen sie aktiviert werden, muss der Benutzer des Automatiken-GUIs immer in Kenntnis der festgeschriebenen Button-Nummern sein. Zum Einstellen der Netzwerk-Nummer des Agenten müssen also Informationsmaterialien über die Vergabe der Button-Nummern hinzugezogen werden (z.B. das Kapitel C.2 der Bedienungsanleitung).

5.2.3 Das Verwenden von gewöhnlichen Agenten im „Opermatik“-Modus bei Versuchstests

Der „Opermatik“-Modus soll in der ersten Testphase ausschließlich mit Automatiken ablaufen, welche nur Netzwerk-Agenten beinhalten. Denkbar wäre allerdings auch eine Testvariante, bei der ein oder mehrere Agenten über die gesamten Streckenabschnitte aktiv sind und nicht vom Operateur angesteuert werden sollen. Deshalb ist es möglich, beim „Opermatik“-Modus Agenten in die Automatik zu integrieren, die keine Netzwerk-Agenten sind. Sie arbeiten selbstständig, sind über die gewählten Streckenabschnitte aktiv und beeinträchtigen nicht die Netzwerk-Agenten.

5.2.4 Die Anzeige von Hinweisen am AMD

Das Anzeigen von Operateurs- oder Agenten-Hinweisen am AMD ist in einigen Punkten nicht ausreichend:

- Die Pfeilsymbole im „Fahrtrichtung und Geschwindigkeit“-Morph des visuellen und auditiven Hinweisemorphs im AMD (siehe Abbildung 2) sind nicht voneinander unterscheidbar. Vergibt der Operateur z.B. einen visuellen Richtungshinweis, so wird dieser zwar im AMD angezeigt, jedoch wird er während des Hinweises nicht informiert, ob dieser auditiv oder visuell ist. Das AMD konnte allerdings aus Gründen der Vergleichbarkeit zu vorherigen Testabläufen nicht optisch verändert werden.
- Das Eingreifen von Agenten in die SAM wird dem Operateur am AMD angezeigt. Bei mehreren gleichzeitigen Agenten-Eingriffen kann derzeit nur über einen Eingriff informiert werden, da das AMD weder optisch noch funktionell verändert werden durfte. In diesem (recht seltenen) Fall tritt ein Informationsverlust für den Operateur auf, da die anderen Agenten-Eingriffe nicht sichtbar sind. Eine mögliche Lösung wäre das Erweitern des AMDs um weitere Anzeigemöglichkeiten.

5.3 Ausblick

Trotz der zahlreichen Erweiterungen und Verbesserungen sind die Aufgaben und Weiterentwicklungsmöglichkeiten des ATEO-Systems mit dieser Arbeit noch nicht abgeschlossen. Neben der Umsetzung des Projektziels sind einige Ideen entstanden, die den Umfang und die Funktionalität des ATEO-Systems in zukünftigen Projekten erweitern könnten. Einige dieser Ideen, wie der visuellen Übersicht der Agenten- und Button-Aktivitäten in Charts, konnten in diesem Projekt bereits umgesetzt werden. Andere, wie der vollständigen Automatisierung der Operateurstätigkeiten am AMD, konnten wegen fehlenden Agenten und zeitlichen Einschränkungen noch nicht umgesetzt werden. Diese Ideen werden hier für eventuell nachfolgende Projekte festgehalten.

5.3.1 Das Fortsetzen der Agenten-Entwicklung

Das Fortsetzen der Agenten-Entwicklung ist für dieses Projekt unumgänglich. Allerdings wurden bisherige Agenten nur auf die automatische Unterstützung der MWB im Automatik-Modus zugeschnitten und nicht für den Gebrauch durch Operateure. Es müssten also Situationen antizipiert werden, in denen der Operateur anhand von komplexen Strategien agiert. Betrachtet man diesen Standpunkt, so könnten ganz neue Ereignisse und Agenten entwickelt werden wie z.B. ein Agent, der unmittelbar vor dem Gabelungsbereich entscheidet, ob den MWBn ein Hinweis zur Befahrung des linken oder rechten Gabelzweigs gegeben wird. Ein solcher Agent konnte in dieser Arbeit zwar imitiert werden, jedoch ist das Entscheidungskriterium (die Position des Tracking-Objekts relativ zur Mittellinie) nicht ausreichend auf das Fahrverhalten (Tendenz oder Vorrausschau der aktuellen Fahrtrichtung) der MWB abgestimmt. Eine andere Möglichkeit wäre es, einen Agenten für harte Eingriffe zu entwickeln, der in bestimmten Streckensituationen, wie dem Durchfahren von Hindernissen, die Geschwindigkeit des Tracking-Objekts einschränkt.

5.3.2 Die Integration von Dominanzverhalten

Die Integration von Dominanzverhalten der Agenten ist auch eine denkbare Erweiterung des ATEO-Systems, mit welcher die Agenten untereinnander in dominante und rezessive Gruppen klassifiziert werden könnten, etwa wie bei einem Ranking. Diese Erweiterung würde den Vorteil mit sich bringen, dass die Kombination mehrerer Ereignisse von unterschiedlichen Agenten abgearbeitet werden könnte, ohne die Wirkungsgrenzen auf diese Ereignisse abstimmen zu müssen (siehe Aktivierungsvoreinstellungen im Ereignissystem: 4.2.3).

5.3.3 Anpassungen im Sinne der Bedienbarkeit des AMDs an den neuen Testmodus „Opermatik“

Um die Vergleichbarkeit der gewonnenen Testdaten verschiedener Modi zu wahren, durften keine optischen Veränderungen am AMD vorgenommen werden. Dies schränkt allerding die Gebrauchstauglichkeit und Bedienbarkeit des AMD ein. Für den neuen „Opermatik“-Modus ist das Setzen des gelben Rahmens bei der Aktivität der Agenten wahrscheinlich nicht ausreichend. Dadurch, dass nur einige Buttons diese Möglichkeit haben, müsste der Operateur einen visuellen Hinweis über die Art des Buttons am Button selbst erhalten.

Literatur

[1] Balzert, H.: *Lehrbuch der Software-Technik: Software-Entwicklung.* 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, 2001

[2] Daum, B.: *Java 6: Programmieren mit der Java Standard Edition.* Addison-Wesley, 2007

[3] Fuhrmann, E.: *Entwicklung eines GUI für die Konfiguration der Software-Komponente zur Systemprozessüberwachung und -kontrolle in einer psychologischen Versuchsumgebung.* Diplomarbeit, Oktober 2010

[4] Grechenig T.; Bernhart, M.; Breiteneder, R.; Kappel, K.: *Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten.* München: Pearson Studium, 2010

[5] Hasselmann, M.: *(Titel unbekannt.)* Diplomarbeit in Vorbereitung

[6] Kosjar, N.: *Die Gebrauchstauglichkeit des Automatiken-GUI im Projekt Arbeitsteilung Entwickler Operateur (ATEO).* Studienarbeit, September 2011

[7] Kosjar, N.: *Ein Ereignis-System für das ATEO Automation Framework sowie die Implementierung und Testung von auditiven und visuellen Hinweisen.* Diplomarbeit, März 2012.

[8] Papert, S.; Minsky, M.: *International MIT memo (1970).* Dreyfus, 1981

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

[10] Seid, A.: *Erweitern der Log-Datei und der Analyse von Log-Dateien im ATEO-Projekt.* Studienarbeit, März 2012

[11] Wandke, H.; Nachtwei, J.: *The different human factor in automation: the developer behind vs. the operator in action.* 2008

[12] Weidner-Kim, H.: *(Titel unbekannt.)* Diplomarbeit in Vorbereitung

A Klassendiagramme

A.1 SAM: SAMControllerAgents

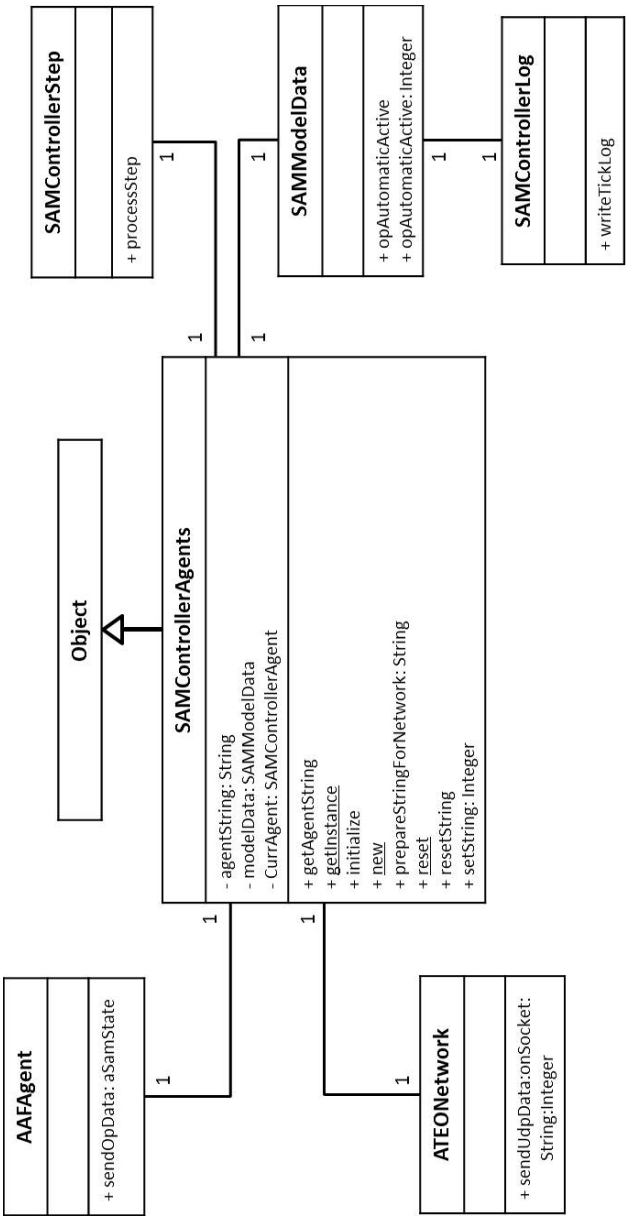


Abbildung 20: *Klassendiagramm SAMControllerAgents; Es werden nur die neu angelegten Methoden und Variablen veranschaulicht.*

A.2 AMD: OPAbstractAutomatic

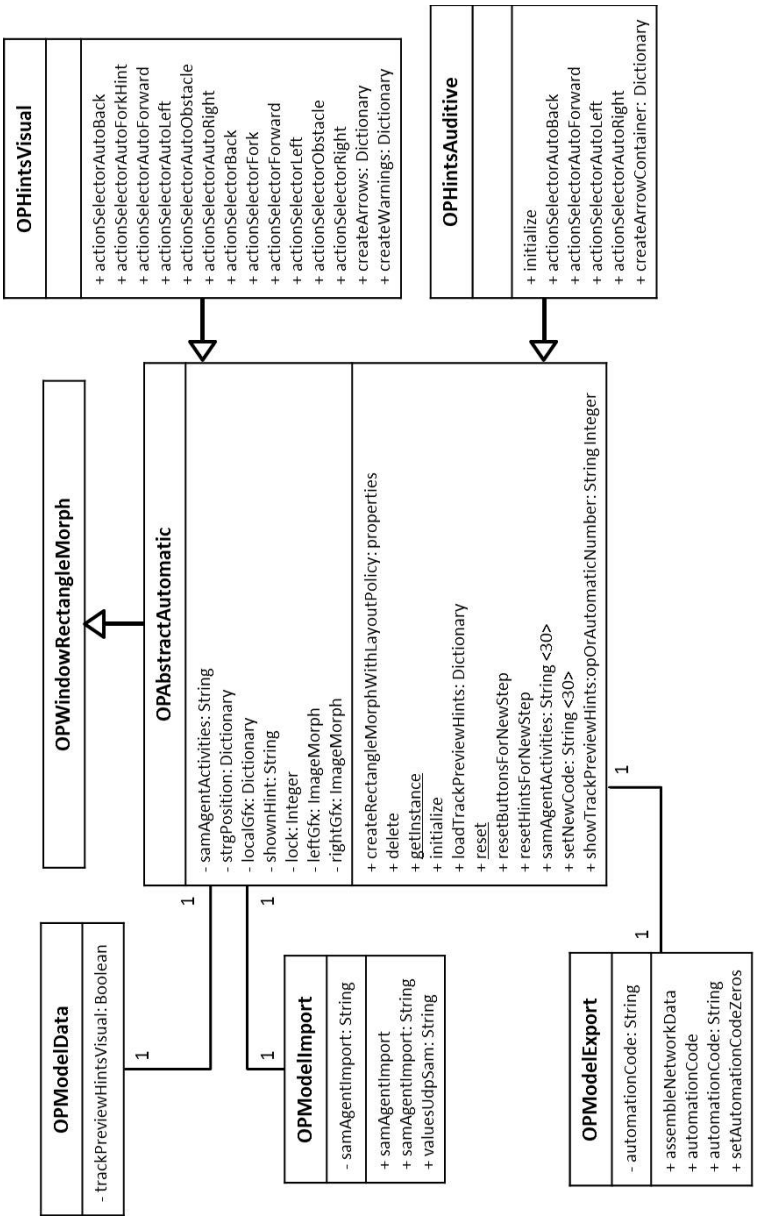


Abbildung 21: *Klassendiagramm OPAbstractAutomatic; Es werden nur die neu angelegten Methoden und Variablen veranschaulicht.*

B Inhalt der DVD

In diesem Abschnitt wird der Inhalt der DVD aufgeschlüsselt. Auf der DVD befinden sich folgende Daten:

- Die Diplomarbeit in schriftlicher Form mit dem Stand der Abgabeversion.
- Die Bedienungsanleitung zur Benutzung und zum Starten des „Opermatik“-Modus als einzelnes Exemplar.
- Das weiterentwickelte Teilmodul der SAM, welches das AAF beinhaltet, liegt im Ordner „ATEO“. Hier befindet sich auch der Unterordner „tools“, in dem die neue LFA-Version 2.0 zu finden ist.
- Das Analysetool LFA in der Version 2.0 befindet sich als einzelne Software separat im Ordner „LFA2.0“.
- Die Software des Operateursarbeitsplatzes für den „Opermatik-Modus“ kann im Ordner „OA3.0“ gesichtet werden. Durch die sehr hohe Auflösung des AMD wurde hierbei auf den Vollbild-Modus verzichtet, um allen Lesern den Zugang zur Software zu ermöglichen.

Es wird darauf hingewiesen, dass diese ATEO-Komponenten zum Zeitpunkt der Fertigstellung dieser Diplomarbeit aktuell sind. Nachfolgende Versionen können auf dem gemeinsamen Assembla-Workspace (`www.assembla.com/spaces/ATEO`) gefunden und heruntergeladen werden.

C Bedienungsanleitung zur Benutzung und zum Starten des „Opermatik“-Testmodus

Dieses Dokument stellt eine Anleitung zum Einbinden und Konfigurieren von Netzwerk-Agenten in eine Automatik und dem anschließenden Starten des „Opermatik“-Modus dar. Als erstes wird für diesen Zweck eine Einführung ins Operateurs-Display und in die Bedienung des Automatiken-GUIs[23] gegeben sowie die Integration der zur Zeit entwickelten Agenten vorgestellt. Im Anschluss werden die genauen Abläufe für das Starten des neuen Testmodus erläutert. Die Auswertung der angefallenen Testdaten mit dem Analysetool sind in dieser Anleitung der letzte Punkt.

C.1 Definitionen

SAM und MWB:
Die Socially Augmented Microworld (SAM) ist das eigentliche Forschungsobjekt, in welcher die Mikroweltbewohner (MWB) ein Tracking-Objekt entlang einer virtuellen Strecke steuern. Automatiken oder ein Operateur können die MWB durch auditive oder visuelle Hinweise unterstützen.

Automatiken-GUI:
Das Automatiken-GUI ist ein Tool der SAM-Software am Rechner der zwei Testpersonen. Mit ihr können Agenten einer Automatik konfiguriert werden.

AMD:
Das ATEO-Masterdisplay (AMD) ist die Benutzeroberfläche des Operateurs. Es beinhaltet eine Streckenvorschau, sowie mehrere Schaltflächen, um die Testpersonen zu überwachen und bei Bedarf kontrolliert einzugreifen.

[23] Graphical User Interface (grafische Benutzeroberfläche) (GUI)

LFA:
Das Logfile-Analysetool (LFA) ist das Werkzeug zum automatisierten Aufbereiten der angefallenen Auswertungsdaten.

C.2 Das AMD im „Opermatik“-Modus

Die aktuelle Version 3.0 des AMD ist mit einigen neuen Schaltflächen in der Lage die Vergabe aller visuellen und Teile der auditiven Hinweise durch Agenten zu automatisieren. Die Arbeitsweise wird im Folgenden beschrieben:

Das alte AMD:
Mit den Buttons der Vorgänger-Version 2.8.4 des AMD ist der Operateur in der Lage, die MWB zu unterstützen oder in ihr Fahrverhalten einzugreifen. Dazu werden ihm Schaltflächen zur Verfügung gestellt, mit denen er z.B. visuelle oder auditive Hinweise für die MWB vergeben oder auch harte Eingriffe am Fahrverhalten der MWB vornehmen kann, wie die Begrenzung der Geschwindigkeit oder der Einflussverteilung der einzelnen MWB am Tracking-Objekt. Dazu kann er mit der linken oder rechten Maustaste die Schaltflächen anwählen, welche dann die gewünschte Aktion ausführen.

Das AMD 3.0:
Die Schaltflächen für die visuellen Hinweise und die auditiven Richtungshinweise wurden überarbeitet und haben nun eine neue Funktion: sie differenzieren zwischen der Anwahl mit der linken oder der rechten Maustaste mit jeweils unterschiedlichen Funktionen. Das Drücken der linken Maustaste führt die gewohnten Aktionen der Schaltflächen aus. Es wird folglich ein Hinweis gegeben oder ins Fahrverhalten eingegriffen. Die rechte Maustaste führt nun zur Aktivierung einzelner durchnummerierter Agenten einer vorher erstellten Netzwerk-Automatik. Dabei wird beim Drücken ein gelber Rahmen um den Button gelegt, welcher anzeigt, ob der oder die Agenten hinter dem Button aktiv sind oder nicht. Im Idealfall imitieren die Agenten, die mit dem Drücken des Buttons aktiviert wurden, genau die Aufgabe, die der Operateur mit die-

ser Schaltfläche ausführt. In Notsituationen, in denen die Agenten und damit der Button aktiv sind, kann der Operateur mit der rechten Maustaste den Agent deaktivieren oder mit dem linken Maus-Button zwei Aktionen ausführen - und zwar das Deaktivieren des Agenten und das gleichzeitige Vergeben des Hinweises.
Die Nummerierung der Agenten innerhalb der Automatik im Automatiken-GUI ist dabei mit den Schaltflächennummern des AMD (siehe Abbildung 22) zu synchronisieren. Diese Nummernvergabe ist statisch, kann aber vom versierten Benutzer im Quellcode angepasst werden.

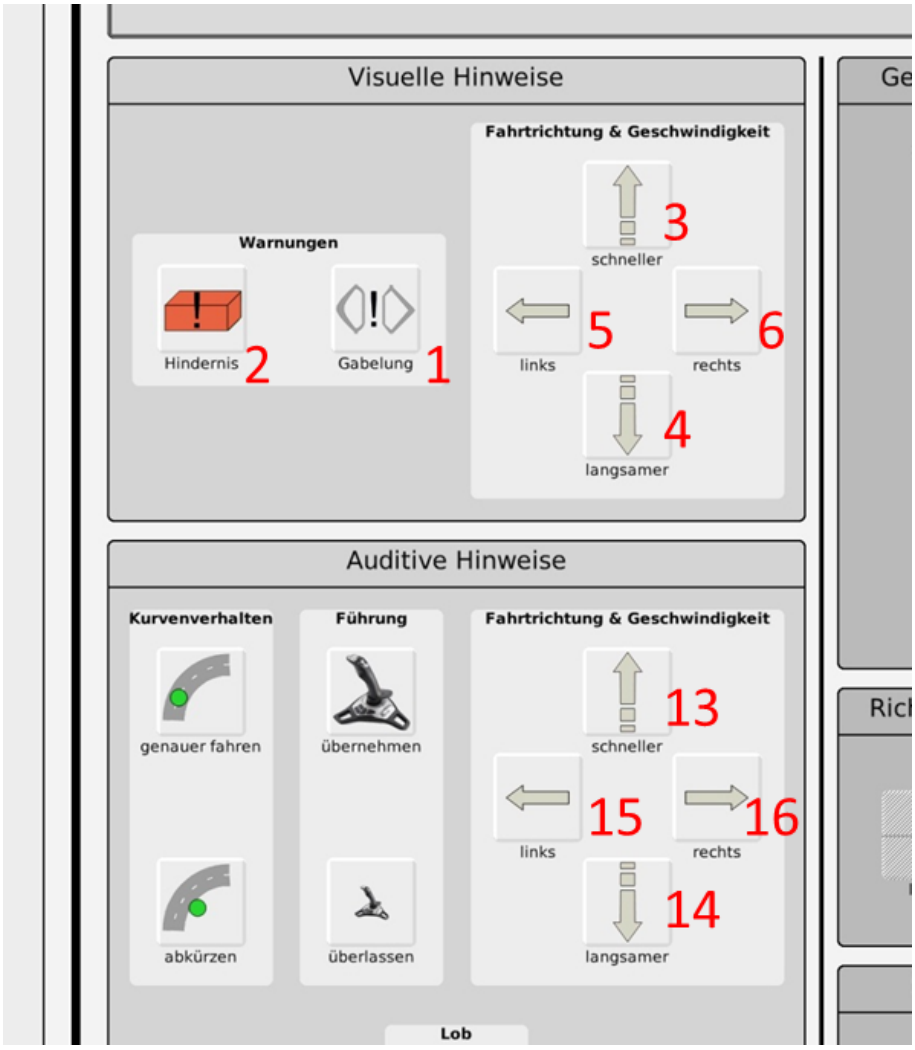


Abbildung 22: *AMD: Nummerierung der Netzwerk-Buttons.*

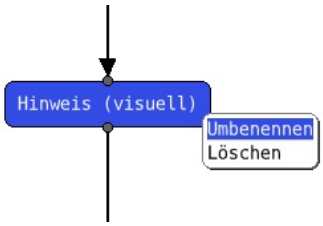
C.3 Das Automatiken-GUI

Das von E. Fuhrmann[3] entwickelte Automatiken-GUI ist einer der zentralen Bestandteile des „Opermatik“-Testmodus. Mit ihm können Agenten von Automatiken erstellt, gespeichert und geladen werden. Für die Erweiterung von Agenten zu Netzwerk-Agenten wurde ein Eigenschaftenfeld integriert, mit dem man den Agenten zum Netzwerk-Agenten definiert und gleich im Anschluss die Nummer einstellt, die synchron zum AMD-Button ist (siehe Abbildung 22). Mit dieser Nummer wird also der Agent im Versuchsverlauf aktiviert, wenn der Button am AMD aktiv ist.

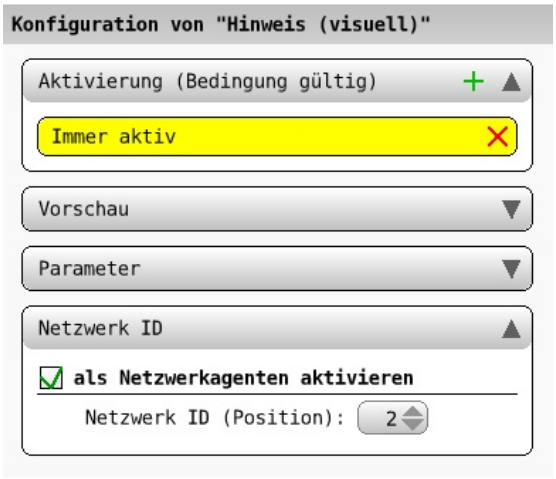
C.4 Das Erstellen einer Automatik

Es folgt nun ein Beispiel für das Erstellen einer Automatik, deren Agent bei Hindernissen warnt:

Beim Öffnen des Automatiken-GUIs befindet sich im linken Bildschirmteil eine Liste aller Agenten, deren Entwicklung abgeschlossen ist. In diesem Beispiel wird ein „visuelle Hinweise“-Agent verwendet und in das mittig gelegene weiße Graphenfenster gezogen. Um den Agenten vollständig in den Graphen zu integrieren, muss er mit der Quelle und der Senke verbunden werden. Dies stellt ein Sinnbild für den Datenverlauf und die Manipulation von Daten durch den Agenten dar.
Durch Anwählen des Agenten mit der rechten Maustaste kann sein Name innerhalb der Automatik geändert oder er aus der Automatik gelöscht werden (siehe Abbildung 23a). Durch das Klicken mit der linken Maustaste wird sein Eigenschaftenbereich an der rechten Seite geöffnet. Es öffnen sich Fenster zum Einstellen des aktivierenden Ereignisses, eine Vorschau und ein Parameterfenster für die Auswahl, Position oder Blinken eines visuellen Hinweisbildes. Als letztes befindet sich im untersten Bereich des Eigenschaftenbereiches das Fenster für das Einstellen des Agenten als Netzwerk-Agent (siehe Abbildung 23b).



(a) Automatiken-GUI: Der Agent im Graphen.



(b) Automatiken-GUI: Die Eigenschaft Netzwerk ID.

Abbildung 23: *Agent im Graphen (a) und Netzwerk-ID (b)*

Der Eigenschaftenbereich „Aktivierung“:
Hier wird das auslösende Ereignis konfiguriert, bei welchem der Agent aktiv wird. Vorerst ist hier das Ereignis „Immer aktiv“ eingestellt. Dieses wird entfernt durch das Anwählen des roten Kreuzes und dann durch ein neues Ereignis ersetzt. Für diesen Zweck können mit Betätigen des grünen „+“-Zeichens ein oder mehrere Ereignisse hinzugefügt werden. Das Ereignis „Bevor/im Hindernisbereich“ erscheint ebenfalls in der Ereignis-Liste und wird durch einen Linksklick der Maus als Ereignis eingestellt. Das Dreieck rechts neben dem Ereignis öffnet seinen Eigenschaftenbereich (siehe Abbildung 24). Hier können nun die obere und untere Grenze (Werte in Pixel relativ zum Hindernis) der Aktivierung festgelegt werden. Standardmäßig sind hierbei die Grenzen -300 und 300, also die Aktivierung des Agenten 300 Pixel vor bis 300 Pixel nach dem Hindernis, eingestellt.

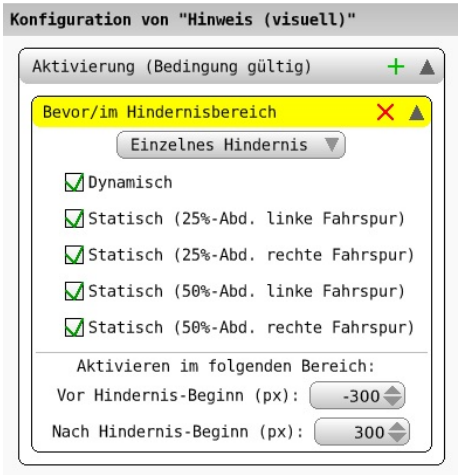


Abbildung 24: *Automatiken-GUI: Konfiguration der Grenzen eines Ereignisses.*

Der Eigenschaftenbereich „Parameter“:
In diesem Konfigurationsbereich wird unter anderem die Art des Anzeigebildes eingestellt. Für das Anzeigen eines Hindernisbildes wird dieses aus der Liste ausgewählt, die sich beim Drücken auf das voreingestellte Bild öffnet. Die Koordinaten der Anzeige können entweder durch das Ziehen des Bildes in der Vorschau an die gewünschte Stelle oder das manuelle Einstellen der x- und y-Koordinaten im SAM-Bildschirm vorgenommen werden.

Der Eigenschaftenbereich „Netzwerk ID“:
Es soll ein Netzwerk-Agent für das automatisierte Anzeigen eines visuellen Hindernishinweises erstellt werden. Dazu wird zunächst der Checkbutton „als Netzwerk-Agenten aktivieren“ gewählt. Nun ist der Agenten bereit, als Netzwerk-Agent zu agieren und der darunterliegende Bereich ist nicht mehr ausgegraut. In diesem kann jetzt die Nummer des Agenten eingestellt werden. Da der Agent als Hinderniswarner eingestellt werden soll, wird die synchrone Nummer zur Button-Nummer des AMD (siehe Abbildung 22), also die Nummer „2“ eingestellt.

Das Abspeichern des Netzwerk-Agenten zu einer Automatik und das Laden in das Automatiken-GUI:
Im oberen Bereich des Automatiken-GUIs kann der gültige Agent mit Drücken des „Speichern“-Buttons in einer Automatik unter einem beliebigen Namen gespeichert werden (im nachfolgenden „AgentOpermatik“ genannt). Dazu wird die Datei „AgentOpermatik.aaf“ automatisch von der Software im ATEO-Verzeichnissystem abgelegt. Für spätere Ergänzungen oder Manipulationen der Automatik-Agenten kann diese durch das Drücken des „Laden“-Buttons in das Automatiken-GUI geladen werden.

Das Integrieren der Automatik in den Versuchsablauf:
Im Ordner „config.tracks“ der Verzeichnisstruktur befindet sich die Datei „steps.txt“. Sie ist das zentrale Glied für den Ablauf des Testdurchgangs und beinhaltet die Nummer der Versuchsabschnitte, die Wahl der Steuerung durch einen oder beide MWB, den Streckenabschnittsnamen und die Einstellung der Hinderniskonfiguration. Soll nun eine Automatik in einen gewünschten Streckenabschnitt integriert werden, so wird diese Textzeile durch ein Semikolon als Separator und anschließend den Namen der Automatik ergänzt (siehe Listing 14).

```
...
 1;  1; lernstrecke;   obstacleConfig_0; AgentOpermatik
 2;  1; testabschnitt; obstacleConfig_0; AgentOpermatik
 3;  2; lernstrecke;   obstacleConfig_0; AgentOpermatik
 4;  2; testabschnitt; obstacleConfig_0; AgentOpermatik
 5; 12; hauptabschnitt_ohne_manipulationen; obstacleConfig_0; AgentOpermatik
 6; 12; hauptabschnitt_ohne_manipulationen; obstacleConfig_0; AgentOpermatik
 7; 12; hauptabschnitt1; obstacleConfig_1; AgentOpermatik
 8; 12; hauptabschnitt2; obstacleConfig_2; AgentOpermatik
 9; 12; hauptabschnitt_lang; obstacleConfig_7; AgentOpermatik
10; 12; hauptabschnitt_lang; obstacleConfig_7; AgentOpermatik
11; 12; hauptabschnitt_lang; obstacleConfig_7; AgentOpermatik
...
```

Listing 14: *Definition der Streckenabschnitte in der Datei steps.txt*

Mit dem Abarbeiten der letzten Schritte sind alle Vorbereitungen für einen Testdurchgang abgeschlossen und der Test kann gestartet werden.

C.5 Vorbereitung für das Starten des „Opermatik“-Modus (AMD)

Um den Versuchstest im „Opermatik“-Modus zu starten, muss am Operateursrechner die Datei „squeak.exe“ des Ordners „OA 3.0“ ausgeführt werden. Das gerade geöffnete Squeak-Fenster beinhaltet einen Morph, also einen Button, mit der Aufschrift „Create GUI v.3.0“. Mit dem Drücken des Buttons folgt ein Konfigurationsfenster (siehe Abbildung 25), mit welchem das AMD für den Operateur individuell zusammengestellt werden kann. Durch das Drücken des „Alle“-Buttons, öffnen sich alle Bedienelemente des AMDs, bis auf die Anzeige der Joystick-Auslenkung der MWB. Diese kann jetzt manuell hinzugefügt werden.

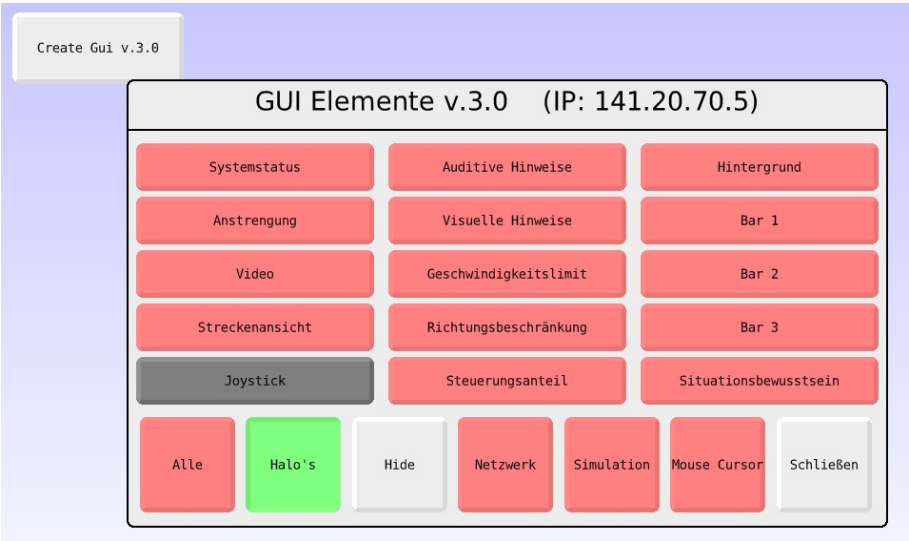


Abbildung 25: *AMD: Fenster zur Konfiguration des AMDs.*

C.6 Vorbereitung für das Starten des „Opermatik“-Modus (SAM)

Zunächst wird die Datei „squeak.exe“ des Ordners „runtime-sam+aaf“ ausgeführt und das bereits bekannte Squeak-Fenster öffnet. Mit der Auswahl des „SAM (User Mode)...“ im ATEO-Menü öffnet sich ein Konfigurationsfenster für das Einstellen einiger Parameter des Versuchsablaufes (siehe Abbildung 26). Darunter befinden sich auch die drei möglichen Assistenzarten Operateur, Automatik und „Opermatik“.

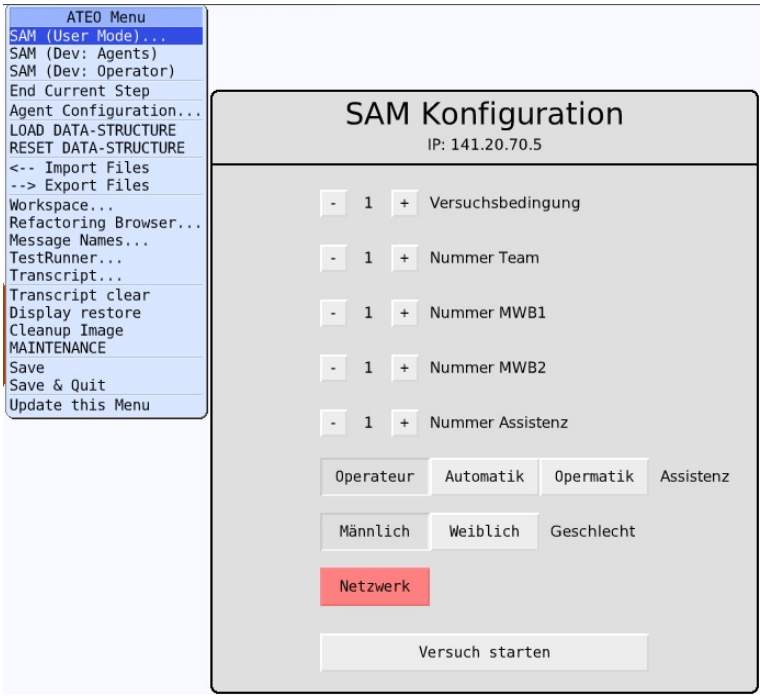


Abbildung 26: *SAM: Fenster zur Konfiguration des Versuchs.*

C.7 Das Starten des „Opermatik“-Modus

Um den Versuch zu starten wird nun am AMD-Menü die Schaltfläche „Netzwerk“ gedrückt, welche im aktiven Zustand seine Farbe in grün ändert. Der Rechner des Operateurs ist also bereit, eine Netzwerkverbindung einzugehen und wartet. Die Schaltfläche „hide“ verursacht das Ausblenden des AMD-Konfigurationsfensters. Dies kann mit Drücken der Leertaste rückgängig gemacht werden. Im Konfigurationsfenster des SAM-Systems wird nun die Assistenzart „Opermatik“ ausgewählt und der rote Button „Netzwerk“ betätigt. Es erscheint ein Eingabe-Fenster in dem die IPv4-Adresse des Operateursrechners eingegeben wird. Bei unveränderter Rechnertopologie kann ebenso der Befehl „ateo1“ (Rechnername) eingegeben werden. Die beiden Rechner sind verbunden, wenn der „Netzwerk“-Button von roter zu grüner Farbe ändert.

Bei fehlerhaftem Verbinden wird eine Fehlermeldung der Squeak-Programmierumgebung gegeben. Diese kann geschlossen werden und beide „Netzwerk“-Buttons sind als inaktiv (rot) einzustellen. Im Anschluss muss dieser Schritt (Abschnitt C.7) wiederholt werden.

Bei erfolgreicher Netzwerkverbindung kann der Testversuch nun am SAM-Rechner gestartet werden.

C.8 Die Auswertung der Versuchsdaten

Für die Auswertung der Versuchsdaten des „Opermatik“-Assistenzmodus oder aber auch der beiden anderen Modi wird die LFA-Version 2.0 sowie die eigentlich auszuwertenden Daten benötigt. Das LFA kann mit Installation des entsprechenden Java Developement Kits aus dem Ordner „dist“ unter dem Dateinamen „LFA_v2.0.jar“ gestartet werden. Unter Datei -> Datei öffnen kann dann der komplette Datensatz eingelesen und anschließend die

Auswertung durch das LFA gestartet werden. Dazu wird der „Start“-Button gedrückt und dann im Feld rechts davon der Fortschritt der Auswertung angezeigt (siehe Abbildung 27). Nachdem das LFA die Daten ausgewertet hat, entsteht ein neuer Ordner im Überordner „results“, welcher das Verzeichnis mit den ausgewerteten Daten beinhaltet. Zu finden sind hier:

- Der Ordner „charts“:
  Er beinhaltet Tabellenbilder, welche die harten Eingriffe durch Agenten aufschlüsselt.
- Der Ordner „OAcharts“:
  Er beinhaltet ein Bild eines Koordinatensystems für jeden Streckenabschnitt, auf welchem die Button- sowie Agenten-Aktivitäten der dreißig Netzwerk-ID-Stellen zu sehen sind. Außerdem wird jede Gabelung durch einen horizontalen roten Strich angezeigt.
- Der Ordner „xls“:
  Hier befindet sich die Microsoft Excel-Datei der kompletten Auswertung.

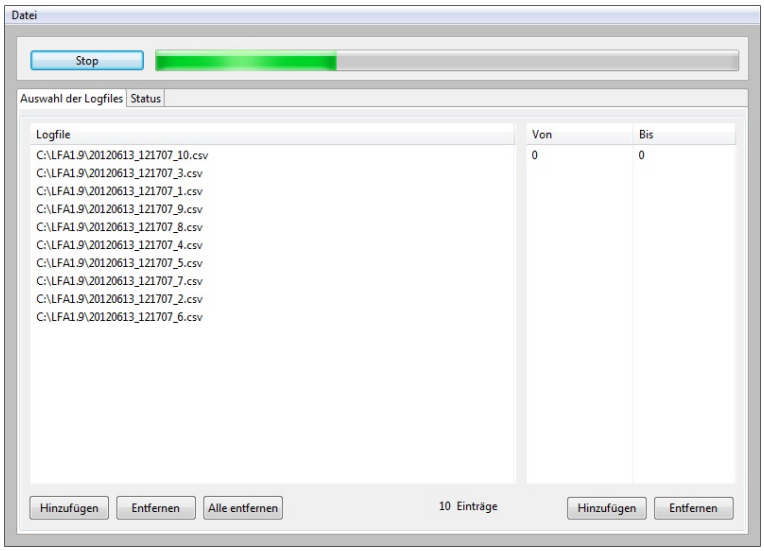


Abbildung 27: *LFA: Abarbeitung eines Bearbeitungsvorgangs.*

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 .....

Statement of authorship

I declare that I completed this thesis on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this nor a similar work has been presented to an examination committee.

Berlin, ...