Humboldt-Universität zu Berlin

Lehr- und Forschungsgebiet Softwaretechnik

Integration von Automatiken in das ATEO-Master-Display (AMD) - Machbarkeitsstudie

Studienarbeit

Von:

Stefan Schulze
Institut für Informatik
Matrikelnummer: 510626
Berlin, 31. Januar 2012

Gutachter:	Betreuer:
Prof. Dr. Klaus Bothe	Dipl.-Inf. Nicolas Niestroj
Prof. Dr. Hartmut Wandke	Dipl.-Psych. Charlotte Meyer

Stefan Schulze: Integration von Automatiken in das ATEO-Master-Display (AMD) - Machbarkeitsstudie, Studienarbeit, © Januar 2012

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 ATEO-Projekt

ATEO (Arbeitsteilung Entwickler-Operateur) 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 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.

Im Versuchsablauf überwacht eine Versuchsperson (der Operateur) ein komplexes dynamisches System, welches durch einen Trackingprozess zweier weiterer Probanden (sogenannte Mikroweltbewohner) simuliert wird, die gemeinsam ein virtuelles Objekt eine Strecke entlang steuern sollen (Dieses Subsystem wird in Kap. 1.1.1 als „Socially Augmented Microworld“ (SAM) näher beschrieben). Um die Komplexität des Prozesses zusätzlich zu erhöhen, werden Konfliktsituationen[1] erzeugt. Die Aufgabe des Operateurs besteht darin, die Trackingleistung zu optimieren, zum Beispiel, indem er standardisierte auditive oder visuelle Hinweise an einen oder beide Mikroweltbewohner (MWB) gibt. So kann der Operateur die MWB auf Konfliktsituationen im Vorhinein aufmerksam machen oder deren Bewältigung im Nachhinein loben. Die Versuchsumgebung ähnelt der eines Überwachungssystems, z.B. der Leitwarte in einem Kraftwerk oder der Flugüberwachung durch Lotsen.

Bisherige Studien untersuchten die Leistung des Operateurs (Effizienz, Effektivität und Sicherheit). Dieser soll zukünftig durch eine Automatik ersetzt und die Automatik-Leistung dann mit der Leistung des Operateurs verglichen werden. Der Entwickler entwickelt die Automatiken für den Vergleich.

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

1.1.1 SAM

In der „Socially Agumented Microworld“ (SAM), einer um eine menschliche Komponente angereicherte Mikrowelt, agieren die Versuchspersonen (MWB) sowie das eigentliche Forschungsobjekt (das virtuelle gesteuerte Objekt). SAM wird demnach durch einen Trackingprozess repräsentiert, bei dem die MWB ein Objekt entlang einer virtuellen Strecke 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 Trackingobjekt so schnell wie möglich ins Ziel zu steuern. Dabei erhalten die MWB jedoch unwissentlich zwei unterschiedliche Instruktionen: 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 empfundenen Anstrengung angeben, welcher anschließend dem Operateur übermittelt wird.

1.1.2 AMD

Das sogenannte ATEO-Master-Display (AMD) dient als Interface für den Operateur und bietet ihm die Möglichkeit, den laufenden Prozess (SAM) zu überwachen und bei Bedarf einzugreifen. Die Eingriffe sind hierbei in harte und weiche Eingriffe zu differenzieren. 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 beinflussen. 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, so z.B. an Gabelungen. Als weiche Eingriffe sind dem Operateur auditive und visuelle Hinweise zur Hand gegeben, um damit den steuernden MWB Hilfestellungen in Hinblick auf ihr gemeinsames Ziel, die schnelle und fehlerfreie Bewältigung der Strecke, zu geben. Außerdem werden dem Operateur verschiedene Möglichkeiten der Überwachung geboten: Zum einen durch ein Videobild der beiden MWB und ihrer Joystickauslenkung, zum anderen durch eine erweiterte Streckenansicht, welche nicht nur den für die MWB sichtbaren Streckenbereich darstellt, sondern auch einen zusätzlichen Teil der als nächstes zu befahrenden Strecke. Diese Vorausschau ermöglicht es dem

[2] insgesamt müssen 11 unterschiedliche Strecken befahren werden

Operateur, mögliche Konfliktsituationen zu antizipieren. 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 angezeigt, der seine Farbe gemäß der gefahrenen Geschwindigkeit anpasst. Detaillierte Informationen dazu können in der Diplomarbeit von Schwarz[3] nachgelesen werden. [2]

1.1.3 AAF

Um die automatisierte Regelung des Trackingprozesses zu untersuchen, wurde ein Automatiken-GUI entwickelt, mit dem man eine oder mehrere Automatiken hinzuziehen kann[4]. Die Schnittstelle zwischen dem Automatiken-GUI und SAM bildet das ATEO Automation Framework (AAF). Zum einen definiert das AAF Agenten nebst Methoden, die 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. [3]

Dieser Verbund kann als Set in der Automatiken-GUI eingelesen, verändert oder auch gespeichert (als *.aaf-Datei) werden. In der Textdatei „Steps.txt“ wird die Reihenfolge der Versuchsabschnitte festgelegt. Hier kann dann für jeden Abschnitt ein Set aus Automatiken mit Hilfe des jeweiligen Set-Namen hinzugefügt werden. Das Aktivieren von Automatiken im Versuch kann unter [3] und eine Bedienungsanleitung der Automatiken-GUI unter [4] nachgelesen werden.

[3] H. Schwarz: Fenster zum Prozess: ein Operateursarbeitsplatz zur Überwachung und Kontrolle von kooperativem Tracking.[1]

[4] Es ist durchaus möglich, eine Automatik direkt auf Programmcode-Ebene in Squeek/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.

1.1.4 ALS

Als ATEO-Lab-System (ALS) wird die Zusammenführung der Frameworks zu einer Versuchsumgebung bezeichnet. Es beinhaltet also das AMD als Operateursarbeitsplatz (OA) und davon räumlich getrennt (schwarzer Balken in Abb. 1) 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 Anstrengungen einem bestimmtem Skalenwert zuordnen soll. Der errechnete Wert wird dem Operateur am Anfang des nächsten Abschnittes am AMD angezeigt.

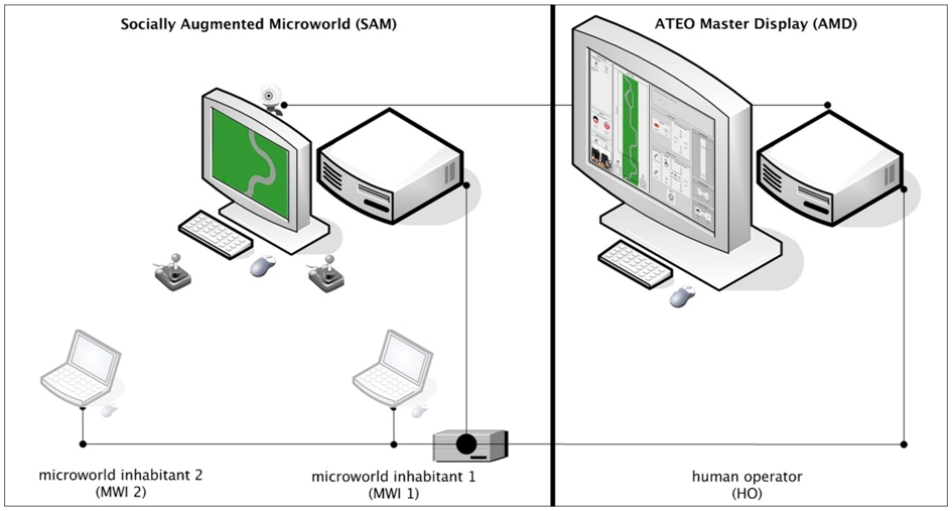


Abbildung 1: *Versuchsaufbau des ATEO-Lab-Systems: links: SAM, rechts: AMD*

1.2 Smalltalk

Die einzelnen Softwarekomponenten von ATEO werden mit Smalltalk und der Programmierumgebung squeak (http://www.squeak.org) entwickelt. Smalltalk setzt die Eigenschaften der Objektorientierung streng um, was sich darin äußert, dass alles ein Objekt ist und dies wie in keiner anderen Sprache auch konsequent umgesetzt ist. Weiterhin sind alle Methoden öffentlich und alle Attribute privat [2].

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

Den Button-Morphen müssen *ActionSelector* zugewiesen werden. Diese sind Instanzmethoden, die ausgeführt werden, wenn die Schaltfläche betätigt wird.

1.3 Ziel der Studienarbeit

Momentan ist es nur möglich, die Versuchspersonen von Automatiken oder von einem Operateur unterstützen zu lassen. Ziel dieser Studienarbeit ist es, ein koorperatives Design für die *Kombination* von Operateur und Automatik als dritten Schritt zu entwickeln. Hierfür werden Schaltflächen in das AMD integriert, die es dem Operateur möglich machen, Automatiken getrennt voneinander hinzuzuschalten, um sich auf andere Aufgabenbereiche zu konzentrieren oder vor Überanspruchung zu schützen. Dazu müssen alle 3 Frameworks (siehe Abb. 2) des ATEO-Systems verstanden und eine Verbindung geschaffen werden, indem die Netzwerkschnittstelle und die Interfaces zwischen den Komponenten erweitert werden.

Zu den Ergebnissen gehören nicht nur die Machbarkeit des Projektes, sondern auch Erkenntnisse über den Umfang, den Zeitaufwand und die Mittel, mehrere Automatiken vom AMD anzusteuern. Außerdem wird untersucht, ob es Hindernisse oder Widersprüche zwischen den aktuellen Kenntnissen und dem Projektziel gibt, welche die weitere Entwicklung oder den Einsatz bestimmter Automatiken einschränken würde. Probleme könnten hier bei der Integration von mehr als nur einer Automatik eintreten oder bei der

Benutzung von mehr als einer Automatik gleichzeitig. Ebenso denkbar ist es, dass es Hindernisse gibt, die eine Neuimplementierung der Frameworks notwendig machen würden und die Machbarkeit aus zeitlichen Gründen ausgeschlossen werden müsste.

Um eine Aktivierung der Automatik ohne Zeitverzögerung zu gewährleisten, ist auch die Frage der Performance interessant. Bei einem optimalen Versuchsablauf sollte möglichst wenig Verzögerung auftreten. Momentan findet ein Paketaustausch alle 40ms statt, was für das An- und Abschalten der Automatik ausreichend ist. Ob durch das zusätzliche Ansteuern der Automatiken Verzögerungen auftreten, muss hier untersucht werden.

1.4 Gliederung

Das *erste Kapitel* stellt die Einführung in die Studienarbeit dar. Hier werden Begriffe und Funktionen der 3 Frameworks und der Programmierumgebung definiert, die für das weitere Verständnis notwendig sind. Weiterhin wird das Ziel der Studienarbeit erläutert.

Im *zweiten Kapitel* wird auf die Problemstellung, die der Eingriff in die 3 Frameworks darstellt, und auf die Definition der Anforderungen eingegangen. Im Anschluss werden die zwei Automatiken und ihre Funktionsweisen vorgestellt.

Das *Kapitel drei* spiegelt den Entwurf und die Implementierungen der Arbeit dar. Dabei werden die Anpassungen der Klassen aller Komponenten beschrieben und auf die Umgestaltung des SAM- und AMD-GUI eingegangen.

Das *Kapitel vier* beinhaltet eine kurze Zusammenfassung über die Ergebnisse und den Erfolg der Arbeiten.

„Diskussion und Ausblick“ ist anschließend das *fünfte Kapitel*, in dem die Studienarbeit kritisch betrachtet und ein Ausblick auf mögliche Anforderungen der Diplomarbeit gegeben wird.

Im *Anhang* sind die Instanz- und Klassenmethoden zu finden, die teilweise editiert oder neu erstellt werden mussten. Sie geben einen Überblick über den Umfang und die Komplexität der Arbeit.

2 Problemanalyse und Definition

2.1 Problemstellung

Um Automatiken in den Operateursarbeitsplatz zu integrieren, müssen sowohl das AMD als auch SAM überarbeitet und die bestehende Netzwerkschnittstelle erweitert werden. Dazu ist es notwendig, dem AMD Schaltflächen hinzuzufügen, über die der Operateur mit den Automatiken inter-agieren kann.

Im Umfeld der Studienarbeit werden Buttons in das AMD integriert, mit denen die Automatiken von 2 Agenten aus dem AAF verbunden werden (siehe 2.2). Später werden diese beiden Automatiken während des Ablaufes vom Operateur einzeln an- und abgeschaltet (siehe Abb. 2).

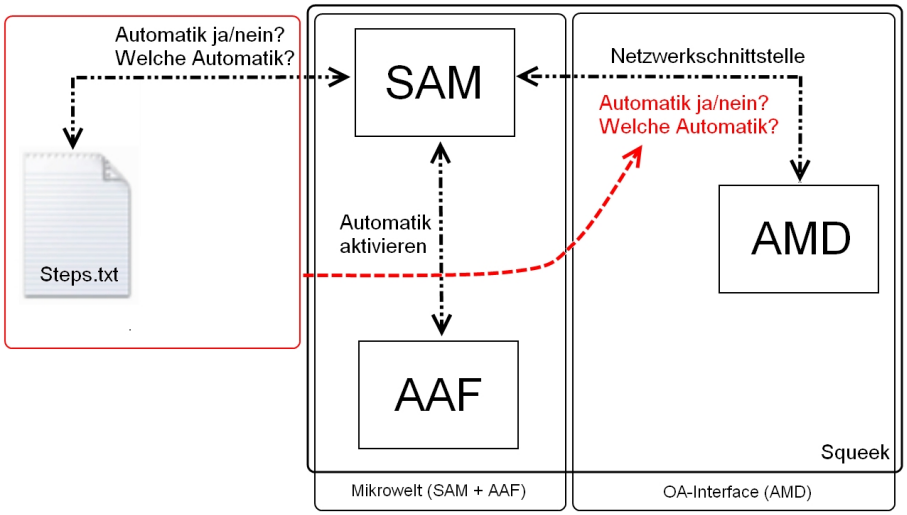


Abbildung 2: *Veränderung des ATEO-Systems: Verlagerung der Automatikaktivierung zur Netzwerkschnittstelle*

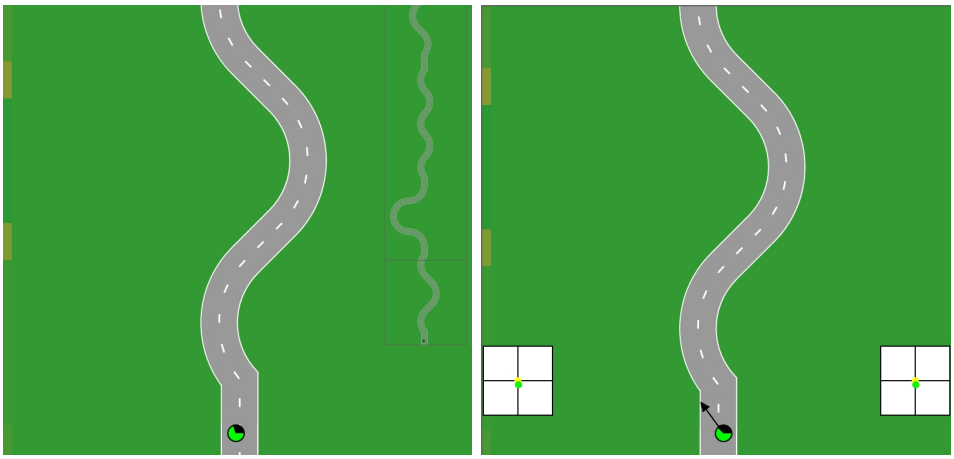
Die notwendigen Optionen laufen dann mit dem bereits bestehenden ATEO-Netzwerkverkehr mit und erzeugen somit keinen zusätzlichen Overhead. Nach dem Informationsaustausch werden die Daten in Echtzeit ausgewertet und die Einstellungen der Automatik angepasst. Die Aufgabe der Textdatei „Steps.txt“ ist nach wie vor, die Reihenfolge der Versuchsabschnitte vorzugeben und optional das Einlesen einer *.aaf-Datei, welche dann die gewünschten Agenten und deren Eigenschaften für die Automatiken enthält.

2.2 Automatikmodule

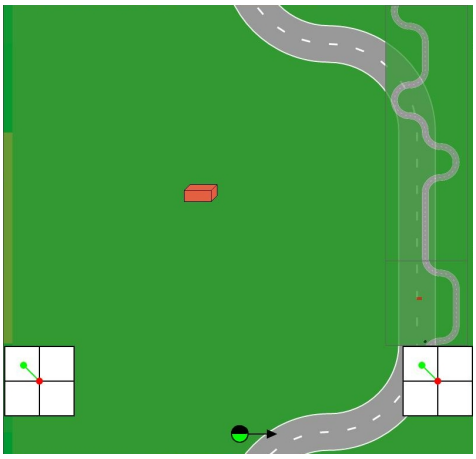
Für das ATEO-System liegen viele Agenten für Automatiken vor, die zum großen Teil von Projektmitgliedern, aber auch in Seminaren entwickelt werden konnten. Um repräsentative Automatiken zu wählen, habe ich mich für die *„Streckenvorschau“* und für eine Automatik entschieden, die als *visueller Helfer für Joystickeingaben* (im Nachfolgenden „visueller Helfer“ genannt) dient (siehe Abb. 3). Diese Automatiken sind deshalb gewählt worden, weil sie fehlerfrei arbeiten und beim Einschalten ein sofortiges Ergebnis liefern, wodurch nicht auf ein aktivierendes Ereignis gewartet werden muss.

Die *„Streckenvorschau“* stellt eine Erweiterung des Interfaces für die MWB dar. Hier erhalten die MWB am rechten Bildschirmrand eine Streckenvorschau, mit der sie etwa vier Abschnitte weit voraussehen können. Diese Vorschau ist transparent, läuft neben dem Tracking mit und ist in der Lage, dynamische und statische Hindernisse anzuzeigen. Damit ist sie, ähnlich wie ein Spurhalte-Assistent, ein Modul, welches die MWB nicht nur unterstützt, sondern auch den Operateur ersetzt.

Der *„visuelle Helfer“* dagegen unterstützt die MWB, indem er jeweils den Joystick-Ausschlag anzeigt und mit einem Pfeil am Tracking-Objekt die Richtung zum Fahrbahnmittelpunkt, also zur optimalen Fahrlinie (racing line), andeutet. Das Design des Trackingobjekts selbst wird dabei nicht verändert. Es ist also ein Assistent, der die MWB bei der Steuerung visuell unterstützt.



(a) Automatik „Streckenvorschau“ (b) Automatik „visueller Helfer“



(c) Beide Automatiken im Zusammenspiel

Abbildung 3: *Automatiken „Streckenvorschau“ und „visueller Helfer“*

3 Entwurf und Implementierung

Im folgenden Kapitel wird auf die Überarbeitung der 3 Frameworks und der Netzwerkkommunikation eingegangen. Dazu wird die Funktionsweise der einzelnen Komponenten und deren Veränderungen erläutert und über die Eingriffe am AMD- und SAM-GUI aufgeklärt.

3.1 Netzwerkkommunikation

Zur Überwindung der räumlichen Trennung erfolgt die Kommunikation zwischen den einzelnen Komponenten über ein Ethernet Netzwerk. Ursprünglich wurde SAM als alleinstehendes Programm entwickelt. Eine Erweiterung um Netzwerkfunktionalität war notwendig, da SAM in der Lage sein muss, mit dem OA Daten über das Netzwerk auszutauschen. Dessen ungeachtet soll sowohl SAM, als auch der OA weiterhin ohne Netzwerkanbindung funktionieren.

Sowohl SAM als auch die Computer zur Messung der Anstrengungen verbinden sich nur mit dem OA. Untereinander tauschen sie keine Informationen aus.

Es wurde eine Basisklasse für SAM und den OA entworfen, von der jede Netzwerkklasse in den einzelnen Komponenten ableitet. Sie implementieren unter anderem Methoden, mit denen Verbindungen aufgebaut und wieder geschlossen und Daten gesendet und empfangen werden können. Durch Parameter können die Methoden individuell in jeder Komponente verwendet werden. [5]

Für die *Kommunikation* vom Operateursarbeitsplatz zu SAM wird in der AMD-Klasse *OPModelExport* ein Netzwerkpaket zusammengesetzt, an den SAM-Rechner geschickt und dort in der Klasse *SAMControllerNetwork* ausgepackt. Beim Zusammensetzen werden durch Token (hier: '#') getrennte Informationen in einen String konkateniert. Dieser String wird alle 40ms ausgewertet und enthält wichtige Informationen, wie zum Beispiel welche visuellen oder auditiven Hinweise angezeigt werden sollen oder ob der Operateur ein Geschwindigkeitslimit erlässt.

Für die *Automatiken-Kommunikation* von SAM und dem AMD wurde dem bestehenden Netzwerkpaket eine weitere Information hinzugefügt. Der *„automationCode“* ist das zentrale Glied der Aktivierung von Automatiken und besteht aus einem zehnstelligen binären Code, genau wie eine 10-Bit-Zahl. Jede dieser Positionen steht stellvertretend für eine Automatik eines Agenten, wobei eine Eins auf einen aktiven und eine Null auf einen inaktiven Agenten hinweist. Da die Studienarbeit eine Machbarkeitsstudie darstellt, sollen hier vorerst zwei Automatiken angesteuert, also auch nur zwei Stellen verändert werden (siehe Tabelle 1). Die verbleibenden acht Positionen des Strings sind momentan nicht belegt.

Position	Name der Automatik	Wert
1	-	0
2	-	0
3	-	0
4	visueller Helfer	0/1
5	-	0
6	-	0
7	-	0
8	-	0
9	-	0
10	Streckenvorschau	0/1

Tabelle 1: *Automatik-Positionen im Netzwerk-String*

Der Wert der Automatiken kann zentral in der Klasse *OPAbstractAutomatic* (siehe 3.2.2) des AMDs an seiner spezifischen Stelle gesetzt und in der Klasse *SAMControllerNetwork* (siehe 3.3.2) für SAM ausgelesen werden. Will man also eine Automatik konfigurieren und hinzufügen, so muss auf beiden Seiten des Kommunikationskanals der String an der gleichen Position verändert und ausgewertet werden, da sonst entweder die falsche oder gar keine Automatik angesprochen wird.

3.2 Objektorientierter Entwurf: AMD

3.2.1 Anpassung der GUI-Elemente

Der Versuchsleiter kann vor Beginn das AMD für den Operateur individuell anpassen. Dafür können am Konfigurator verschiedenste GUI-Elemente ausgewählt und hinzugefügt werden. Dieser wurde als Ganzes vergrößert und um eine vierte Spalte erweitert, die sich ausschliesslich mit Schaltflächen für Automatiken befässt (siehe Abb. 4).

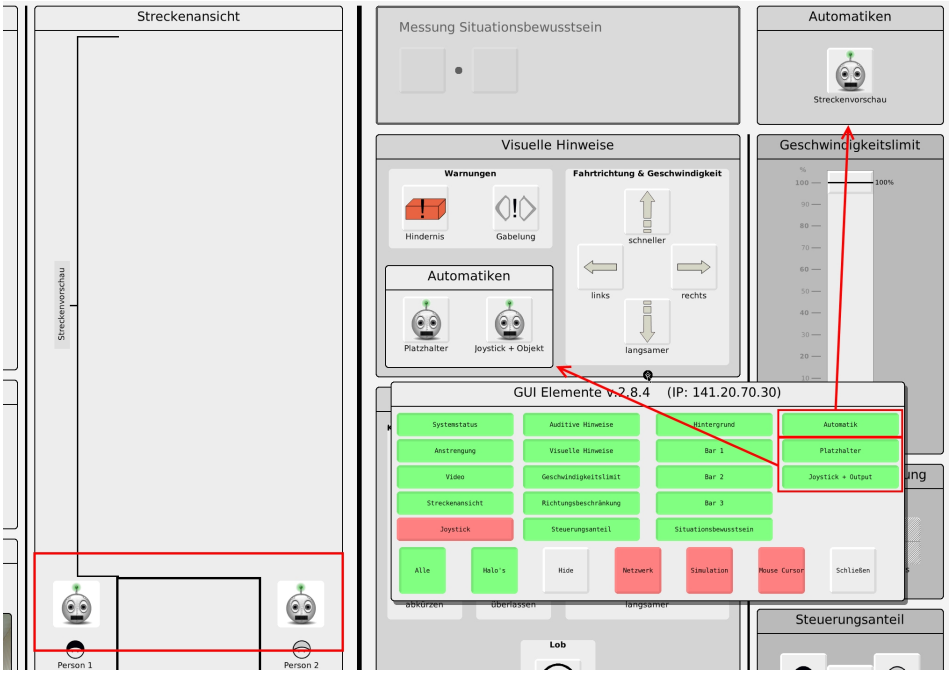


Abbildung 4: *GUI-Elemente und AMD-GUI. Links unten: Einblendung des Automatik-Symbols bei Benutzung einer Automatik; Rechts: Integration der Automatiken-Morphs im AMD durch Bedienen von Buttons im GUI-Elemente*

Mit dem „Automatik“-Button ist es nun möglich, einen „Automatiken“-Morph zu erzeugen, der den Button „Streckenvorschau“ enthält. Ist der Morph hinzugefügt, so wechselt die Schaltfläche standardmäßig von inaktiv (rot) zu aktiv (grün). Die Funktion der Schaltfläche „Alle“, mit welcher alle Morphs aktiviert und deaktiviert werden können, wurde um den „Automatik“-Morph erweitert.

Weiterhin wurden 2 Schaltflächen hinzugefügt, welche zu Beginn ausgegraut sind. Eine der beiden ist eine Automatik zur visuellen Unterstützung, die andere ein Testbutton. Beide Buttons sind allerdings Bestandteil eines Submorphs des „Visuelle Hinweise“-Morphs. Daher können diese Optionen nur gewählt werden, wenn dieser aktiv ist und angezeigt wird. Aktiviert man nun eine der Schaltflächen, so wird ein Submorph in den bestehenden „Visuelle Hinweise“-Morph integriert, der je nach Auswahl einen der beiden oder beide Buttons enthält. Der „Platzhalter“-Button wurde zu Testzwecken der Machbarkeitsstudie integriert und ist zur Zeit ohne Funktion. Mit ihm sollte untersucht werden, ob es möglich ist, genau diese strukturierten Bedingungen für das Erscheinen des Submorphs zu erzeugen. Mit dem „Joystick + Output“-Button kann während des Versuchs die Automatik „visueller Helfer“ aktiviert werden, welche den MWBn die Joystickposition und Abweichung zum Streckenmittelpunkt anzeigt (siehe Abb. 3). Sind beide Schaltflächen oder der Morph „Visuelle Hinweise“ inaktiv, so bleibt der Submorph verborgen.

3.2.2 Anpassung der Klassen

Für die Umsetzung der Anforderungen (siehe Kapitel 2) wurden drei neue Klassen entwickelt und einige Instanzmethoden der Klasse *OPGuiMasterControl*, *OPHintsVisual*, *OPModelExport* und *OPWindowRectangleMorph* erweitert. Die neuen Klassen sind *OPAutomatic*, *OPAbstractAutomatic* und *OPHintsVisualAutomatic*, welche der Kategorie ATEO-OP-GUI angehören. Die Einordnung der neuen in die bestehenden und überarbeiteten Klassen kann dem Klassendiagramm in der Abbildung 5 entnommen werden.

In der Klasse *OPGuiMasterControl* mussten einige Methoden angepasst und neu erstellt werden:
Um das Platzieren neuer Buttons in einer viertel Spalte zu ermöglichen, wurde in der initialize-Methode die Breite des GUI-Elemente-Morphs um 300 Pixel erhöht. Die Methode initializeButtons wurde um die Erzeugung von drei Schaltflächen im GUI-Konfigurator erweitert, welche je einen neuen ActionSelector zugeteilt bekommen haben. Jedoch unterscheiden sich die Arten der Schaltflächen. Die Automatik-Schaltfläche wurde in der Methode initializeDictionarys hinzugefügt und ist ein sogenannter Switch-Button, welcher bei Aktivierung die Farbe ändert. Die Schaltflächen „Platzhalter“ und „Joystick + Output“ sind ebenso Switch-Buttons, jedoch

mit zusätzlicher „Lock“-Eigenschaft. Dadurch ist es möglich, diese zu sperren und erst unter bestimmten Voraussetzungen zu entsperren. Der Lock-Status wird für jeden der beiden Buttons einzeln in den Methoden setDummybuttonState: und setVisHelpbuttonState: verwaltet. Ist der „visuelle Hinweise“-Button aktiv, so werden die Locks im actionSelectorHintsVisual entsperrt.

Die Klasse *OPAbstractAutomatic* erbt von der Klasse OPWindowRectangleMorph, beinhaltet die Unterklassen OPAutomatic und OPHintsVisualAutomatic und stellt Methoden für deren grundlegende Funktionen bereit. So ist sie dafür zuständig, ein Symbol bei Aktivierung einer Automatik einzublenden und den Code des Netzwerkstrings an einer spezifischen Stelle in der Methode setNewCode: zu bearbeiten. Das Symbol wird im Morph „Streckenansicht“ an der Stelle eingeblendet, an welcher der Operateur auch die aktiven Hinweise für die MWB sieht. Allerdings ist es rezessiv, wird also bei Einblendung der anderen Symbole in deren Anzeigezeitraum überlagert. Um die genaue Position der Automatik im String zu bestimmen, beinhaltet diese Klasse einen „Dictionary“. Dieser besteht aus einer Liste, in der man jedem der Einträge einen Wert oder eine Eigenschaft zuweisen kann. Hier wird jeder Automatik eine gewünschte Postition im automationCode zugewiesen. Diese Stelle muss äquivalent zur Auswertungsmethode auf der SAM-Seite sein, da sonst entweder kein oder der falsche Agent einer Automatik angesteuert wird.

Die Klasse *OPAutomatic* wird aufgerufen, wenn die Schaltfläche „Automatik“ im GUI-Konfigurator aktiviert wird. Sie erzeugt einen Morph, der den Button „Streckenvorschau“ enthält. Wird dieser betätigt, so wird der Automatik-Code an der Stelle zehn geändert (siehe Tabelle 1) und das Automatik-Symbol eingeblendet (siehe Abb. 4). Dies hat zur Folge, dass die Automatik für die MWB aktiviert wird.

Ein Objekt der Klasse *OPHintsVisualAutomatic* wird in der Klasse OPHintsVisual erzeugt und dient dazu, den Submorph, welcher die beiden Buttons für die visuellen Automatiken beinhaltet, zu erstellen oder auszublenden. Da diese Klasse, wie oben beschrieben, eine Subklasse von OPAbstractAutomatik ist, kann auch sie den Netzwerkstring bearbeiten und Symbole für aktive Automatiken einblenden.

Die Klasse *OPHintsVisual* wird bei Erzeugung des „visuelle Hinweise“-Morphs gestartet und erzeugt den Submorph für die visuellen Automatiken. Seine Methode automaticVisibility:onOrOff: editiert die Button-Bibliothek und blendet den Submorph ein, wenn visuelle Automatiken in der GUI-Elemente ausgewählt werden. Mit seinen Methoden visHelpVisibility: und dummyVisibility: können dann die beiden Buttons einzeln angezeigt werden.

In der Klasse *OPModelExport* wurde die Instanzmethode „automationCode“ als Getter- (zum Abrufen) und Setter-Methode (zum Manipulieren) für die Bearbeitung der gleichnamigen Instanzvariable hinzugefügt, die wiederum von der Methode assembleNetworkData der gleichen Klasse den Netzwerkcode übergeben bekommt. Hierauf greift dann die Klasse OPAbstractAutomatic zu, welche diesen Code manipuliert.

OPWindowRectangleMorph verwaltet als Super-Klasse die Bibliothek „guiElementPositions“, die die Positionen der Morphs im AMD enthält. Da sich der Submorph für die visuellen Automatiken am Morph „visuelle Hinweise“ orientiert, musste hier nur „OPAutomatic“ hinzugefügt werden.

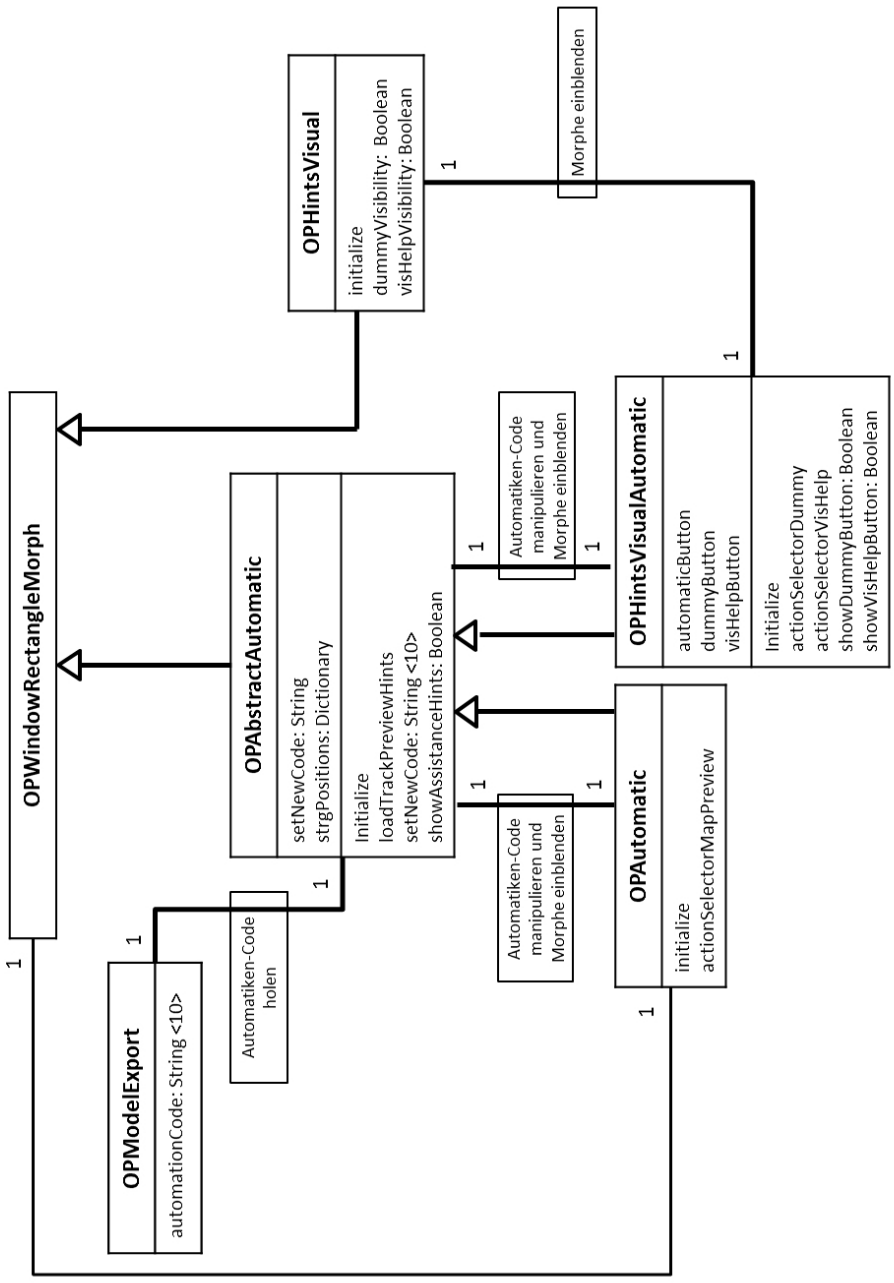


Abbildung 5: *Klassendiagramm - relative Einordnung der neuen Klassen des Operateursarbeitsplatzes*

3.3 Objektorientierter Entwurf: SAM

3.3.1 Anpassung des GUI-Elements

Um SAM zu konfigurieren, wurde eine GUI entwickelt (siehe Abb. 6), mit der verschiedene Parameter des Versuchs voreinstellbar sind. Damit kann der Versuchsleiter unter anderem bestimmen, ob das Experiment durch eine Automatik oder einen Operateur überwacht wird.

Um den neuen Versuchsmodus in das SAM-GUI einzupflegen, wurde ein neuer Button eingeführt. Die Auswahl der „Opermatik“-Assistenz (*Oper*ateur + Auto*matik*) sorgt dafür, dass die Automatiken eingelesen werden (wie im Automatik-Modus). Das Betätigen des „Netzwerk“-Buttons ist anschließend dafür zuständig, dass eine Verbindung zum Operateursarbeitsplatz aufgebaut wird (wie im Operateur-Modus). Voraussetzung dafür ist, dass die Netzwerkverbindung zuvor am AMD-GUI aktiviert wurde.

Abbildung 6: *SAM-Konfigurator mit „Opermatik“ als Assistenzauswahl*

3.3.2 Anpassung der Klassen

Der „Opermatik-Button“ wurde in der Methode createGuiElements der Klasse *SAMViewGuiConfig* integriert und der Konfigurationsmorph auf 500Pixel verbreitert.
Weiterhin wurde ihm der *actionSelectorMachineHumanAssistance* zugewiesen, welcher die gleichen Funktionen hat, wie der *actionSelectorMachineAssistance*. Diese Methode sorgt dafür, dass die voreingestellten Automatiken über die Datei „Steps.txt“ ausgewählt und schliesslich eingelesen werden. Die Quellcode-Doppelung wurde in Kauf genommen, um eine klare Trennung der 3 Assistenzmodi gewährleisten und diese im weiteren Projektverlauf besser bearbeiten zu können.

Die Methode processNetwork der Klasse *SAMControllerNetwork* wurde so angepasst, dass die neue, zwölfte Information eingelesen und an die Methode processAutomationCode: der gleichen Klasse weitergegeben wird. In dieser wurde für jede Automatik eine Klassenvariable angelegt, die eine eindeutige Position im Netzwerkstring zuweist.

In der Klasse *SAMModelData*, erhält jede Automatik zwei Instanzvariablen, die den Aktivierungszustand der Automatik als Boolean speichern und wiedergeben. Sie werden mit jeweils zwei Setter- und Getter-Methoden bearbeitet und so auch zeitgleich im AAF-Graphen an- bzw. abgeschaltet (siehe 3.4.1).

3.4 Objektorientierter Entwurf: AAF

3.4.1 Anpassung der Klassen

Eine Schnittstelle für die Kommunikation der beiden Frameworks SAM und AAF stellt *SamState* dar, mit der die Agenten des AAF den Trackingprozess in SAM beeinflussen können. Die compute-Methode der Klasse *AAFNode* ist dafür zuständig, dass die compute-Methoden der im Automatik-Graphen enthaltenen Agenten aufgerufen werden, welche die eigentlich auszuführenden Funktionalitäten jeder Automatikfunktion beinhaltet. Diese liefern dann je nach Agent einen veränderten *SamState* zurück.

Mit Hilfe der für jeden Agenten eindeutigen Instanzvariable in SAM wird nun hier das Aufrufen der compute-Methoden zugelassen oder unterdrückt. So ist es möglich, dass das Hinzuschalten einer Automatik ohne Zeitverzögerung von statten geht, da ihr Agent trotzdem vor Testbeginn initialisiert wird. Wird der Agent unterdrückt, so liefert diese Methode einen unveränderten SamState zurück und ist damit als Automatik deaktiviert.

Es ist auch möglich, einen Agenten für eine Automatik mitlaufen zu lassen, welcher nicht vom AMD an- oder abgeschaltet werden kann. Dazu muss er nur Teil der *.aaf-Datei des aktuellen Versuchsabschnittes sein.

4 Ergebnisse

In der Studienarbeit konnte gezeigt werden, dass es möglich ist, mindestens zwei voneinander unabhängige Automatiken in den Operateursarbeitsplatz zu integrieren und sie mit einem einfachen Mechanismus dem Operateur zur Verfügung zu stellen. Dafür konnte die Netzwerkschnittstelle sinnvoll erweitert und die Frameworks SAM und AAF erfolgreich editiert und ergänzt werden. Durch das Eingreifen in alle Frameworks ist der Umfang und der Zeitaufwand der Implementierungen hoch, da eine Vielzahl von Programmierern auch eine Vielzahl von Programmierstilen und -strukturen mit sich führt. Probleme bei der Integration einer oder mehrerer Automatiken sind allerdings nicht aufgetreten. Werden den Versuchspersonen Automatiken hinzugeschaltet, so tritt keine Verzögerung im Trackingprozess auf.

Es kann also nun auch die Kombination von Operateur und Automatik im ATEO-Projekt untersucht werden. Die beiden Automatiken können leicht durch andere, für den Versuch nützlichere Automatiken ausgetauscht werden.

Durch die Entwicklung des neuen Modus und den Abschluss dieser Studienarbeit, schlage ich ein Versionsupdate für den Operateursarbeitsplatz auf die Version 3.0 (vorher 2.8.4) und für SAM auf die Version 2.5 (vorher 2.3.4) vor.

5 Diskussion und Ausblick

5.1 ATEO

In der Diplomarbeit sollen, zusätzlich zu den 2 bestehenden Automatiken, die vom Operateursarbeitsplatz angesteuert werden können, weitere Automatiken hinzukommen. Dazu wird ermittelt, welche Agenten bereits ausgereift sind und den Operateur unterstützen können. Die Automatik „Streckenvorschau“ wird mit hoher Wahrscheinlichkeit verwendet werden, wobei der „visuelle Helfer“ für den späteren Gebrauch eher ungeeignet ist. Für die genauen Anforderungen an den Umbau des ATEO-Lab-Systems wird in Zusammenarbeit mit Nicolas Niestroj und Charlotte Meyer ein Lastenheft erarbeitet.

5.1.1 Netzwerkstring

Um das gesamte Framework so generisch wie möglich zu programmieren, wäre es sinnvoll, dem „automationCode“ (momentan 10 Stellen lang) eine variable oder ausreichend große Länge zuzuweisen. Ein variable Länge könnte hilfreich sein, damit die Netzwerkpakete so klein wie möglich bleiben, um die Belastungen im LAN gering zu halten. Allerdings stellt sie bei der Auswertung und Differenzierung der Agenteninformationen eine komplizierte Handhabung dar. Auf der anderen Seite würde eine feste, große Länge eine bessere Gruppierung in die Aufgabenbereiche ermöglichen (z.B.: 0-9 als visuelle, 10-19 als auditive Automatiken). Dies würde aber gleichzeitig eine Einschränkung der Menge an Automatiken nach sich ziehen.

Generell soll mit der Vergrößerung sichergestellt werden, dass auch eine Vielzahl von Informationen für eine Vielzahl von Automatiken übertragen werden kann. Ebenso denkbar ist es, die Informationen nicht nur mit Einsen und Nullen zu kodieren um den Informationsgehalt jeder Automatik zu erhöhen; jedoch müssen vorerst die Art und Komplexität der Agenten feststehen.

5.1.2 AMD

Für das AMD werden in der Diplomarbeit weitere Submorphs entwickelt, die die Automatiken je nach Aufgabenbereich gruppieren. So könnte es zum Beispiel einen „auditive Automatiken“-Morph geben, in dem alle denkbaren auditiven Automatiken aktiviert werden könnten. Welche Automatiken im Einzelnen zum Einsatz kommen und wo sie im AMD platziert werden, wird vorher in einem Workshop ermittelt. Dabei werden ebenso ingenieurpsychologische Richtlinien von Charlotte Meyer in die Architektur einfliessen, die gute Usability gewährleisten sollen, indem sie zum Beispiel ein Überladen verhindern. Das gilt auch für die Überarbeitung des Konfigurators GUI-Elemente, der die Buttons für die Erzeugung der Morphs enthält. Hier muss entschieden werden, ob noch eine fünfte Spalte für die Automatikmodule eingerichtet wird.

Momentan bekommt der Operateur nur angezeigt, ob Automatiken aktiv sind oder nicht. Später wird ein Informationspanel entwickelt, welches dem Operateur explizit anzeigt, welche Automatik gerade aktiviert ist. Denkbar wäre hier eine Auflistung der Automatiken, denen sich farbliche Kreise anschliessen, die nach dem Ampelprizip arbeiten (grün = aktiv; rot = inaktiv) oder eine farbliche Umrahmung der Buttons nach gleichem System.

Dadurch, dass jeder Automatikbutton des AMDs das gleiche Symbol hat, wird die Usability für den Operateur beeinträchtigt. Hier wird das Design der Buttons individualisiert und optimiert, indem die Buttons für jede Automatik ein eigenes aussagekräftiges Bild erhalten. Außerdem wird bei der Namensgebung dieser Buttons darauf geachtet, dass sie genau wie die anderen Elemente eine kurze deutschsprachige Bezeichnung bekommen, sodass der Button „Joystick + Output“ zu „visuelle Hilfe für Joystickbewegungen“ umbenannt werden könnte.

Eine weitere Variante für das Einbinden von Automatiken in das AMD wäre, sie direkt an die vorhandenen Buttons zu binden. So könnte die Automatik, die vor Hindernissen warnen soll, mit dem Button aktiviert werden, den der Operateur schon jetzt benutzt, um manuell auf Hinweise aufmerksam zu machen. Für die Trennung von manuellen Eingriffen des Operateurs und der Aktivierung einer Automatik könnten die Maustasten hilfreich sein (linke Maustaste: manueller Hinweis, rechte Maustaste: Automatik aktivieren). Diese Variante macht es zum anderen möglich, die bewährte Übersicht des AMD beizubehalten.

5.1.3 SAM

Auf der SAM-Seite wird für die nachhaltige, generische Nutzung ein Dictionary von Agenten erstellt, mit dem man neue Automatiken einfach und übersichtlich hinzufügen kann. Denkbar wäre hier ein Array, welches jeder Automatik einen Namen, eine Position im String und einen Zustand zuweisen kann, ohne neue Variablen anzulegen.

Außerdem sollen die Logfiles der Testdurchläufe so angepasst werden, dass genau ablesbar ist, wann und welche Automatiken der Operateur hinzuzieht. Diese Anpassungen sollen dann durch den actionSelector-MachineHumanAssistance angestossen werden.

5.1.4 AAF

Zur Zeit wird ein Agent entwickelt, welcher nicht nur eine Automatik darstellt, sondern nach kurzem Konfigurieren die Aufgaben aller auditiven und visuellen Automatiken annehmen kann. Sollte dieser Agent zeitnah fertiggestellt werden können, so wird er im Zuge der Diplomarbeit eingesetzt. Da dieser Agent allerdings mehrere Aufgabenbereiche abdecken kann, müsste man ihn dem Agentengraphen mit verschiedenen Konfigurationen mehrmals zufügen. Dies wiederum würde meiner Differenzierung der Agenten nach Klassennamen widersprechen. Das AAF müsste also um die Unterscheidung nach Graph-Objektnamen erweitert werden. Dies würde auch den Wegfall der beiden Getter- und Setter-Variablen in der AAF-SAM-Schnittstelle SamState für die Steuerung der Automatiken bedeuten. Dadurch braucht der Nutzer nicht mehr ins AAF eingreifen, um seine Automatiken für den Operateur zugänglich zu machen.

Literatur

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

[2] N. Niestroj: *Erweiterung des ATEO-Systems zur Komplexitätserhöhung in SAM.* Diplomarbeit, März 2008

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

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

[5] C. Leonhard: *Fenster zum Prozess: Weiterentwicklung eines Operateursarbeitsplatzes im Projekt Arbeitsteilung Entwickler Operateur.* Studienarbeit, Juli 2010

A Instanzmethoden

A.1 Instanzmethoden AMD

A.1.1 OPGuiMasterControl

METHODE	BESCHREIBUNG
initializeButtons	Erweiterung um die Erzeugung von 3 Schaltflächen im GUI-Konfigurator mit der Zuweisung von jeweils einem ActionSelector.
initializeDictionarys	Automatik-Morph wurde den Dictionarys hinzugefügt und erhält somit einen Anzeigezustand, Button-Namen und ActionSelector.
actionSelectorHintsVisual:	Wird bei Betätigen der Schaltfläche „visuelle Hinweise“ gestartet und gibt seinen Status jetzt an setDummybuttonState: und setVisHelpbuttonState: weiter.
actionSelectorDummy:	Wird bei Betätigen der Schaltfläche „Platzhalter“ gestartet und gibt seinen Status an die Methode dummyVisibility: der Klasse OPHintsVisual weiter.
actionSelectorVisHelp:	Wird bei Betätigen der Schaltfläche „Joystick + Output“ gestartet und gibt seinen Status an die Methode visHelpVisibility: der Klasse OPHintsVisual weiter.
setDummybuttonState:	Setzt den Lock-Status der Schaltfläche „Platzhalter“ in Abhängigkeit der Aktivierung vom „visuelle Hinweise“-Morph.
setVisHelpbuttonState:	Setzt den Lock-Status der Schaltfläche „Joystick + Output“ in Abhängigkeit der Aktivierung vom „visuelle Hinweise“-Morph.

Tabelle 2: *Instanzmethoden der Klasse OPGuiMasterControl*

A.1.2 OPAbstractAutomatic

METHODE	BESCHREIBUNG
initialize	Initialisiert ein Objekt dieser Klasse und wird automatisch gestartet. Legt außerdem ein Dictionary an, der jeder Automatik eine Stelle im Netzwerkstring zuordnet.
loadTrackPreviewHints:	Stellt Morphs für die Anzeige des Automatik-Symbols bereit.
setNewCode:	Lädt den aktuellen Netzwerk-String, untersucht und ändert ihn an gewünschter Stelle.
showAssistanceHints:	Blendet Morphs bei Benutzung einer oder mehrerer Automatiken ein.

Tabelle 3: *Instanzmethoden der Klasse OPAbstractAutomatic*

A.1.3 OPAutomatic

METHODE	BESCHREIBUNG
initialize	Initialisiert ein Objekt dieser Klasse und wird automatisch gestartet. Erzeugt den Automatik-Morph in der AMD-GUI.
actionSelectorMapPreview	Wird gestartet bei Betätigen des „Streckenvorschau“-Buttons. Veranlasst die Anzeige des Automatik-Symbols und der Netzwerkstringänderung in der Klasse OPAbstractAutomatic.

Tabelle 4: *Instanzmethoden der Klasse OPAutomatic*

A.1.4 OPHintsVisualAutomatic

METHODE	BESCHREIBUNG
initialize	Initialisiert ein Objekt dieser Klasse und wird automatisch gestartet. Erzeugt den „visuelle Automatiken“-Submorph, welcher die beiden Buttons „Platzhalter“ und „Joystick + Output“ beinhaltet.
actionSelectorDummy	Wird gestartet bei Betätigen des „Platzhalter“-Buttons. Veranlasst die Netzwerkstringänderung in der Klasse OPAbstractAutomatic. Ist aber noch ein Dummy-Button, da keine Automatik an- oder abgeschaltet wird.
actionSelectorVisAydan	Wird gestartet bei Betätigen des „Joystick + Output“-Buttons. Veranlasst die Anzeige des Automatiksymbols und der Netzwerkstringänderung in der Klasse OPAbstractAutomatic.
showDummyButton:	Setzt den visible-Status des „Platzhalter“-Buttons auf „true“ oder „false“.
showVisHelpButton:	Setzt den visible-Status des „Joystick + Output“-Buttons auf „true“ oder „false“.

Tabelle 5: *Instanzmethoden der Klasse OPHintsVisualAutomatic*

A.1.5 OPHintsVisual

METHODE	BESCHREIBUNG
initialize	Initialisiert ein Objekt dieser Klasse und wird automatisch gestartet. Erzeugt den Submorph für visuelle Automatiken in der AMD-GUI und einen Dictionary, der die Sichtbarkeit der Buttons des Submorphs verwaltet.
dummyVisibility:	Mit dieser Methode wird im Dictionary die Sichtbarkeit des „Platzhalter“-Buttons und des Submorphs gesetzt.
visHelpVisibility:	Mit dieser Methode wird im Dictionary die Sichtbarkeit des „Joystick + Output“-Buttons und des Submorphs gesetzt.
automaticVisibility:onOrOff:	Überprüft, ob einer der Buttons sichtbar ist und setzt anschließend den Submorph auf sichtbar.

Tabelle 6: *Instanzmethoden der Klasse OPHintsVisual*

A.2 Instanzmethoden SAM

A.2.1 SAMViewGuiConfig

METHODE	BESCHREIBUNG
createGuiElements	Erstellt den Konfigurationsmorph und wurde um den „Opermatik“-Morph erweitert. Diesem wurde der actionSelectorMachineHumanAssistance zugeteilt.
actionSelectorMachineHumanAssistance	Sorgt für die Erhebung und Senkung der Assistenz-Buttons und vergibt den Assistenz-Modus.

Tabelle 7: *Instanzmethoden der Klasse SAMViewGuiConfig*

A.2.2 SAMControllerNetwork

METHODE	BESCHREIBUNG
processNetwork	Teilt das Netzwerkpaket und verteilt die Daten an seine Instanzmethoden. Hier wird auch der Netzwerkstring an die Methode processAutomationCode: weitergereicht.
processAutomationCode:	Wertet die Daten des Automatik-Strings aus und setzt die Variablen der Automatiken auf „true“ oder „false“.

Tabelle 8: *Instanzmethoden der Klasse SAMControllerNetwork*

Zu den Instanzmethoden der Klasse *SAMControllerNetwork* kommen noch die 2 Klassenmethoden mapPreview und visAydan, die jeweils die Position der Automatik im Netzwerkstring zurückgeben.

A.2.3 SAMModelData

METHODE	BESCHREIBUNG
initialize	Initialisiert ein Objekt dieser Klasse und wird automatisch gestartet. Setzt die Instanzvariablen mapPreviewActivated und visHelpActivated auf „true“.
mapPreviewActivated:	Setter der Instanzvariable mapPreviewActivated.
mapPreviewActivated	Getter der Instanzvariable mapPreviewActivated.
visHelpActivated:	Setter der Instanzvariable visHelpActivated.
visHelpActivated	Getter der Instanzvariable visHelpActivated.

Tabelle 9: *Instanzmethoden der Klasse SAMModelData*

A.3 Instanzmethoden AAF

A.3.1 AAFNode

METHODE	BESCHREIBUNG
compute:	Übergibt den aktuellen SamState an die compute-Methoden der Agenten im Graph oder unterdrückt dies, je nach Name und Aktivierung des Agenten.

Tabelle 10: *Instanzmethoden der Klasse AAFNode*

Erklärungen

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Berlin, 31. Januar 2012